Proceedings
Work-in-Progress Session

of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'10)

August 23-25, 2010
Macau SAR, P.R.C.

Edited by Thomas Nolte

Message from the WiP Chair

Dear Colleagues,

Welcome to Macau and to the Work-in-Progress (WiP) Session of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'10). I am pleased to present 9 excellent WiP papers that describe innovative research contributions from the broad field of real-time and embedded systems and applications. The 9 accepted papers were selected from 14 submissions. These proceedings are also published as a Technical Report from Mälardalen Real-Time Research Centre (MRTC), Västerås, Sweden (available at www.mrtc.mdh.se).

The preliminary purpose of the WiP Session is to provide researchers with an opportunity to discuss their evolving ideas and gather feedback from the real-time and embedded systems and applications community at large. The presentation session is limited in duration, and can only provide a brief overview of each WiP paper. I hope however that members of the audience will find ideas presented particularly interesting and want to participate in longer discussions with the authors during the poster session that follows.

I would like to thank the WiP Program Committee members, listed below, for their hard work in reviewing the papers.

Moris Behnam	MRTC/Mälardalen University, Sweden
Marko Bertogna	Scuola Superiore Sant'Anna, Pisa, Italy
Liliana Cucu	INRIA Nancy Grand Est, France
Arvind Easwaran	CISTER/IPP Hurray, Research Group ISEP/IPP, Porto, Portugal
Insik Shin	KAIST, Korea

Special thanks also goes to Eduardo Tovar, Robert I. Davis and Tei-Wei Kuo for their support and assistance.

Thomas Nolte
Work-in-Progress Chair
16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'10)

Table of Contents

Towards Adapting Non-Standard System Execution Traces for Validating Enterprise DRE System QoS Properties

T. Manjula Peiris and James H. Hill
Dept. of Computer and Information Science
Indiana University-Purdue University Indianapolis
Indianapolis, IN USA
Email: {tmpeiris, hillj}@cs.iupui.edu

***Abstract*—System execution traces are useful artifacts for validating enterprise distributed real-time and embedded (DRE) system quality-of-service (QoS) properties, such as end-to-end response time, throughput, scalability, and reliability. With proper planning during development phase of the software lifecycle, it is possible to ensure such traces contain the required properties to simply their analysis for QoS validation. In some case, however, it is *hard* to ensure system execution traces contain the necessary properties, such as with externally developed DRE system components. Consequently, this makes it *hard* to analyze such system execution traces for validation of QoS properties.**

This work-in-progress paper provides two contributions for analyzing system execution traces for enterprise DRE system QoS validation. First, this presents a methodology called SETAF for adapting non-standard system execution traces for analysis of QoS properties. Secondly, this paper presents preliminary results from applying SETAF to externally developed applications and analyzing its QoS properties. Initial results show that is possible to analyze non-standard system execution traces for validate QoS properties without modifying the applications existing source code.

***Keywords*-QoS validation, non-standard system execution traces, adaptation, patterns, dataflow**

I. Introduction

Current trends and challenges. System execution traces [3], [7], [8], *i.e.*, a collection of log messages, are useful artifacts for analyzing soft real-time enterprise distributed real-time and embedded (DRE) quality-of-service (QoS) properties, such as scalability, throughput, and end-to-end response time. A benefit of using system execution traces for QoS validation is that they provide a comprehensive view of the system's behavior and state throughout its execution lifetime as opposed to a single snapshot of the system at a given point in time, such as a global snapshot [10] that can be hard to analyze such concerns throughout an enterprise DRE system's execution lifetime. Likewise, they provide DRE system developers and testers with a rich set of data for analyzing data trends associated with a QoS given property, *i.e.*, how a given QoS property changes with respect to time, such as viewing how latency changed over the system's execution lifetime or points in the execution lifetime where end-to-end response time missed its deadline.

The UNITE [5], [6] tool describes a methodology for validating QoS properties using system execution traces. UNITE applies relational database theory [1] and dataflow models [2], [9] to analyze different QoS properties. UNITE uses these techniques because most system execution traces describe different but related events which happen in different points in time with some keywords. For example, a system execution trace may contain messages that dictate the sending/receiving of an event along with a timestamp for each occurrence of the message. By searching for the send/receive message keywords and using a dataflow model to show their relation, it is possible to mine the system execution trace for such messages and analyze event latency within a enterprise DRE system. More importantly, such analysis can take place irrespective of system complexity, composition, and implementation because the dataflow model is at a higher level-of-abstraction than the concrete system and system execution traces are platform-, technology-, and language-independent artifacts.

Although it is possible to validate QoS properties via system execution traces, the system execution traces must contain several properties, such as identifiable keywords (as described above). Moreover, the dataflow model, which is used to analyze the system execution trace, must also contain several properties, such as identifiable log message formats, *i.e.*, a regular expression that represents similar log messages, and unique relations between different log formats. If planned early enough in the software lifecycle, it is possible to ensure such properties exist in both the dataflow model and system execution trace. Unfortunately, it is not possible to always ensure system execution traces from enterprise DRE systems contain the required for analysis and QoS validation, such as with externally developed software components and systems. It is therefore critical to develop methodologies that will enable non-standard system execution traces (and their dataflow model) to undergo analysis and QoS validation.

Solution approach → Adapt system execution traces using patterns. The adapter software design pattern [4] is a pattern that enables one object to adapt to the expected interface that another object expects. More importantly, this pattern does not require modification of the two original objects. In the context of analyzing non-standard system execution traces to validate enterprise DRE system QoS properties, the adapter pattern can be used to *adapt* the dataflow model and system execution trace to contain the required properties for QoS validation. The main challenge, however, is determining how either the system execution trace and dataflow model must be adapted so that either contains the necessary properties to support analysis and QoS validation.

This work in progress paper therefore presents the initial work on *System Execution Trace Adaptation Framework (SETAF)*, which is framework used to adapt system execution traces and dataflow models so they contain the required properties for analysis and QoS validation. DRE system developers and testers use SETAF by first analyzing the system execution trace to identify the pattern for adapting their system execution trace. They then specify the adaptation pattern as a high-level adaptation specification. UNITE then uses SETAF and adaptation specification to inject the required properties at run-time into the analysis of the system execution trace. This process ensures the system execution trace analysis is valid. Initial results for applying SETAF to an open-source software project show that SETAF is able to adapt non-standard system execution traces without requiring modification to the original source code that generates the system execution trace.

Paper organization. The remainder of this paper is organized as follows: Section II provides a brief overview of UNITE; Section III presents the initial design and methodology of SETAF; Section IV presents the preliminary results for applying SETAF to an open-source project; and Section V presents concluding remarks and future research directions.

II. Brief Overview of UNITE

UNITE is a methodology and tool for analyzing system execution traces and validating QoS properties. DRE system developers and tester use UNITE by first generating a system execution trace what consists of a set of log messages. For example, Listing 1 illustrates a portion of a system execution trace generated by a enterprise DRE system.

```
activating Sensor...
...
Sensor sent message A.1 at 2345
Config received message A.1 at 2347
...
Sensor sent message A.2 at 2376
Sensor sent message A.2 at 2379
Config received message A.2 at 2378
...
Shutting down system...
```

Listing 1. Portion of an example system execution trace.

As shown in Listing 1, the send and receive messages can be used to calculate event latency. To perform such analysis using UNITE, DRE system developers and testers identify what log messages they want to extract using a *log format*. This log format captures both the static and variable portions of the log messages. More importantly, the variables identify what portion of the log message to extract for usage in QoS validation. For example, Listing 2 highlights the log formats for the send and receive messages in Listing 1, respectively.

```
LF1: Sensor sent message {STRING type}.
{INT msgid} at {INT sent}

LF2: Config received message {STRING type}.
{INT msgid} at {INT recv}

Relation:
  LF1.type = LF2.type
  LF1.msgid = LF2.msgid
```

Listing 2. Dataflow model for analyzing system execution trace.

After defining the log formats for extracting metrics of interest from a system execution trace, DRE system developers and testers then define a dataflow model that captures the relationship between the different log formats. This is necessary because enables reconstruction of execution flows in the system (1) irrespective of system complexity and composition and (2) without a need for a global clock to ensure causality [10] because the relations between the log formats preserve causality.

```
AVG(LF2.recv - LF1.sent)
```

Listing 3. Expression for analyzing event latency using UNITE.

Finally, DRE system developers and testers define an expression that validates a given QoS property based on the variables in the log format. For example, Listing 3 highlights the expression evaluating average event latency. UNITE then uses the dataflow models and expression to mine the system execution trace and evaluate the provided expression. Likewise, if the aggregation function (*i.e.*, `AVG`) is removed from the expression, then UNITE will present the data trend for the QoS property undergoing analysis.

III. The Design and Functionality of SETAF

This section describes the current design and functionality of SETAF. This section also uses concrete examples to illustrate concepts realized in SETAF.

A. Challenges Associated with Analyzing Non-standard System Execution Traces

Section II provided a brief overview of UNITE and its technique for analyzing system execution traces to validate enterprise DRE system QoS properties. In order to ensure such validation can occur, however, it is necessary that the values in each relation is unique. For example, as shown in Listing 1, the relation between the event ids is *always* unique. If the relations between log formats is not unique, then there is high probability that the analysis will yield incorrect results.

```
Started doing task A at 12.00
Finished doing task A at 12.01
Started doing task A at 12.02
Finished doing task A at 12.03
```

Listing 4. Portion of a system execution trace that contains a non-standard dataflow model.

For example, Listing 4 illustrates an example system execution trace where the dataflow graph will not have unique relations between the log format. This is because it is *hard* to know start/finish messages are associated with one another without human intervention. Moreover, when an example similar to the one present in Listing 4 is analyzed by UNITE, it will yield incorrect results because it is *hard* to determine correct causality between similar log messages.

With proper planning early in the software lifecycle, it is possible to ensure generated system execution traces have unique relations to facilitate proper analysis. This, however, is not alway possible—especially when analyzing system execution traces generated by third-party systems and their components. Although such non-standard system execution traces may not contain unique relations, the existing relations can be exploited (or adapted) to enforce a unique relation. For example, in Listing 4, although the relation is not unique, it can be adapted to be a unique relation by adding an id to each log message. This will ensure that UNITE analyzes the dataflow model and evaluates the expression correctly. The next section therefore explains how SETAF enables such adaptation of non-standard system execution traces.

B. On Adapting Non-Standard System Execution Traces

As explained in Section III-A, dataflow models that do not contain unique relations for analyzing system execution traces can be adapted to contain unique relations. Unfortunately, it is not possible to adapt each non-standard dataflow model in the same manner to enforce a unique relation. This is because the dataflow model is associated with the given system that generates the system execution trace is used to analyze. A dataflow model therefore can only be reused for different executions of the same system.

Because of this fact, DRE system developers and testers use SETAF by first manually analyzing the non-standard system execution trace. Through this analysis, the DRE system developers and tester identify an *adaptation pattern* for adapting the dataflow model to contain unique relations. More specifically, the adaptation pattern contains details about what variables (and values) needed to be added to each log format (and log message) to enforce a unique relation between each log format that does not contain a unique relation.

Using the example present in Listing 4, DRE system developers then define the adaptation pattern specification that is used by SETAF to adapt the system execution trace. For this example, each log message that represents a start task is proceeded by a finish task before another start task occurs. Using this domain knowledge of the system execution trace, Listing 5 highlights the adaptation pattern specification DRE system developers and testers write to ensure proper analysis of a non-standard system execution trace.

```
Columns:
  LF1.uid, LF2.uid

Init:
  let i = 0;

On LF1:
  LF1.uid = i;

On LF2:
  LF2.uid = i ++;
```

Listing 5. Example of an adaptation pattern specification in SETAF.

As illustrated in Listing 5, first DRE system developers and testers specify what data points need to be added to each log format, *e.g.*, `uid` for `LF1` and `LF2`. DRE system developers and testers then define the initial state of the adaptation pattern. Finally, they define how to adapt each log format (and log message) so that the relations in the dataflow graph are unique. In this example, the `uid` variable is assigned the current value of `i` in both LF1 and `LF2`. In `LF2`, however, the state variable `i` is incremented. This will ensure the next occurrence of `LF1` is differentiated from the previous occurrence of `LF1`, as well as `LF2`. Finally, UNITE analyzes the system execution trace and uses SETAF to adapt its analysis of the system execution trace at run-time to ensure valid analysis, and reconstruction of the different execution flows.

IV. Preliminary Results for Applying SETAF to Apache ANT

This section presents preliminary result for applying SETAF to an example application that contains non-standard system execution traces.

A. Experimental Setup

Section III discussed SETAF's technique for adapting non-standard system execution traces for QoS validation. To determine initial validity of SETAF's technique, we applied SETAF to several Java-based open-source applications, *e.g.*, ANT, Tomcat Web Server, and ActiveMQ JMS Broker. We selected Java-based applications because most standard Java-based applications use log4j (http://logging.apache.org/index.html) to generate system execution traces. It is therefore possible to use UNITE's log4j appender to intercept log messages and store them in a database that is analyzable by UNITE.

One such open-source application that we have analyzed is ANT (http://ant.apache.org), which is a build engine primarily used to build Java applications. To setup the experiment, we first executed ANT to generate a system execution trace. Next, we analyzed the generated system execution trace to identify an adaptation pattern. After we identified the adaptation pattern, we defined an adaptation pattern specification for SETAF. Finally, we used UNITE and SETAF to analyze ANT's generated system execution trace. All experiments were conducted on a Intel core 2 Duo 2.1 GHz processor, with 3GB memory and running 32-bit Windows Vista operating system.

B. Experimental Results

Table I highlights the results for using UNITE to analyze a system execution trace generated by ANT without applying SETAF. As illustrated in this table it is correlating `startTime` and `finishTime` of different ANT tasks. For example if we take the second row of the table an ANT Task named "property" has started at 1500 and finished at 1704. The problem with this table is for some entries (*e.g.*, first and third rows) the `startTime` is greater than the `finishTime`. This is because of the non-uniqueness in the dataflow model used to reconstruct in the dataset from the non-standard system execution trace. Because of the non-uniqueness in the dataflow model used to reconstruct in the dataset from the non-standard system execution trace, it resulted several negative values for the evaluation time of different ANT tasks as illustrated below in Table II.

TABLE I
TABLE RECONSTRUCTED BY UNITE WITHOUT ADAPTATION PATTERN SPECIFICATION.

startTime	LF1.task	finishTime	LF2.task
1500	property	860	property
1500	property	1704	property
1516	available	1511	available
1516	available	1518	available

TABLE II
RESULTS FOR ANALYZING RECONSTRUCTED TABLE IN UNITE WITHOUT ADAPTATION SPECIFICATION.

Task	Time (msec)
available	-630.333333333333
delete	0.0
macrodef	140.0
mkdir	-25.125
path	297.0
patternset	-9.76923076923077
property	-241.4
Total evaluation time (sec)	0.345873

Using the adaptation specification defined for ANT, which is similar to the one illustrated in Listing 5, it is possible to improve the results presented in Table I and Table II. Table III therefore highlights the dataset reconstructed by UNITE after using SETAF to apply the adaptation pattern to the reconstruction process. As shown in this table, `startTime` and `finishTime` are now correlated correctly because of the unique id added by SETAF. In this table, all the values of `LF1.startTime` is not greater than the `finishTime`. So it is a correct mapping table.

TABLE III
IMPROVED TABLE RECONSTRUCTION USING SETAF AND UNITE.

LF1.uid	LF1.task	startTime	LF2.uid	LF2.task	finishTime
1	property	766	1	property	860
2	property	1500	2	property	1704
3	available	1500	3	available	1511
4	available	1516	4	available	1518

Likewise, Table IV illustrates the updated final results for analyzing task execution time after adapting the UNITE's analysis at runtime using SETAF. As shown in this table all the evaluation times for different ANT tasks have positive values, which is the expected analysis results.

V. CONCLUDING REMARKS

This work-in-progress paper presented initial work on SETAF, which is a technique and tool that adapts non-standard system execution traces for QoS validation. SETAF operates by applying adaptation patterns to system execution traces to ensure correct analysis. Preliminary results from applying SETAF to an open-source project highlight that it is possible to perform such adaptation at run-time without modifying the original source code to ensure the generated system execution traces contain the necessary properties for QoS validation. Future research will focus on applying this technique to other applications—including large-scale enterprise DRE systems—to further validate the technique.

TABLE IV
FINAL RESULTS FOR ADAPTING UNITE'S ANALYSIS USING SETAF.

Task	Time (msec)
available	93.6666666666667
delete	55.0
macrodef	79.0
mkdir	2.0
path	390.0
patternset	6.0
property	17.975
Total evaluation time (sec)	0.59851

REFERENCES

[1] P. Atzeni and V. D. Antonellis. *Relational Database Theory*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.
[2] J. T. Buck. A Dynamic Dataflow Model Suitable for Efficient Mixed Hardware and Software Implementations of DSP Applications. In *Proceedings of the 3rd International Workshop on Hardware/software co-design*, pages 165–172, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
[3] F. Chang and J. Ren. Validating System Properties Exhibited in Execution Traces. In *Proceeding of the 22^{nd} IEEE/ACM International Conference on Automated Software Engineering*, pages 517–520, New York, NY, USA, 2007. ACM.
[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
[5] J. H. Hill. Data Mining System Execution Traces to Validate Distributed System Quality-of-Service Properties. In D. A. S. Kumar, editor, *Knowledge Discovery Practices and Emerging Applications of Data Mining: Trends and New Domains*. IGI Global, 2010.
[6] J. H. Hill, H. A. Turner, J. R. Edmondson, and D. C. Schmidt. Unit Testing Non-functional Concerns of Component-based Distributed Systems. In *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation*, Denver, Colorado, April 2009.
[7] N. Joukov, T. Wong, and E. Zadok. Accurate and Efficient Replaying of File System Traces. In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, pages 25–25, 2005.
[8] D. Narayanan. End-to-end Tracing Considered Essential. In *Proceedings of High Performance Transactions Systems*, September 2005.
[9] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and P. Wohed. On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling. In *Proceedings of the 3rd Asia-Pacific Conference on Conceptual modelling*, pages 95–104, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.
[10] M. Singhal and N. G. Shivaratri. *Advanced Concepts in Operating Systems*. McGraw-Hill, Inc., New York, NY, USA, 1994.

Calculating an upper bound on the finishing time of a group of threads executing on a GPU: A preliminary case study

Gurulingesh Raravi and Björn Andersson
CISTER-ISEP Research Center
Polytechnic Institute of Porto
4200-072 Porto, Portugal
ghri@isep.ipp.pt, bandersson@dei.isep.ipp.pt

***Abstract*—Graphics processor units (GPUs) today can be used for computations that go beyond graphics and such use can attain a performance that is orders of magnitude greater than a normal processor. The software executing on a graphics processor is composed of a set of (often thousands of) threads which operate on different parts of the data and thereby jointly compute a result which is delivered to another thread executing on the main processor. Hence the response time of a thread executing on the main processor is dependent on the finishing time of the execution of threads executing on the GPU. Therefore, we present a simple method for calculating an upper bound on the finishing time of threads executing on a GPU, in particular NVIDIA Fermi. Developing such a method is non-trivial because threads executing on a GPU share hardware resources at very fine granularity.**

I. Introduction

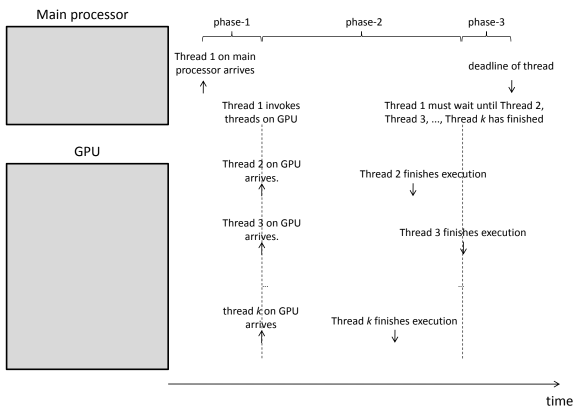
Graphics processors were originally used only for graphics but they have evolved significantly during the recent decade as witnessed by the following. First, graphics processors double their performance every 6 months [1, page 1]; this should be compared with normal CPUs which double their performance every 18 months [1, page 1]. Consequently, graphics processors today offer significantly higher peak performance than a normal CPU. The most recent graphics processor, NVIDIA Fermi [2], has a peak computing performance of approximately one Teraflop [3]; this is approximately thousand times greater than a normal single-core processor in a normal PC. Second, graphics processors are able to perform general-purpose computations, programmed using CUDA APIs [1] with C; hence enabling ”normal software developers” to use graphics processors for data-parallel programs. Therefore, today this type of processor can be thought of as a multicore processor; it has come to be called General Purpose Graphics Processor Unit (GPGPU) or simply GPU (the phenomenon is called GPU computing [2]).

So far, the GPU has been marketed as a ”supercomputer-at-your-desktop” but we believe that its use will also spread to embedded computer systems. A GPU however has no I/O capability and was not designed to run a normal operating system and therefore, a GPU is used as a co-processor to a CPU – the term CPU/GPU computing signifies this.

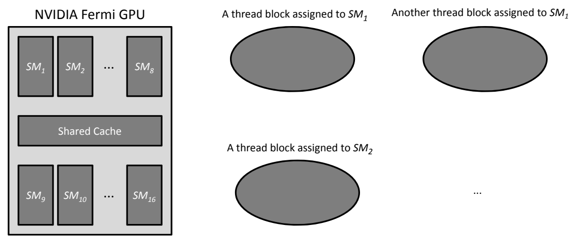
Figure 1(a) shows an example of the use of a GPU as a co-processor. A thread on the main processor arrives and performs some computations (for example reading sensors) and copies data from main processor's memory to GPU's memory. Then the thread on the main processor invokes the threads on GPU and suspends itself. The threads on the GPU execute in parallel on different data that they have been assigned and when these threads finish their execution, the thread on the main processor resumes execution. It copies the data from GPU's memory to main processor's memory and then uses this result (for example for actuation).

We can see that in order for CPU/GPU computing to be possible for hard real-time applications, three research problems must be solved:

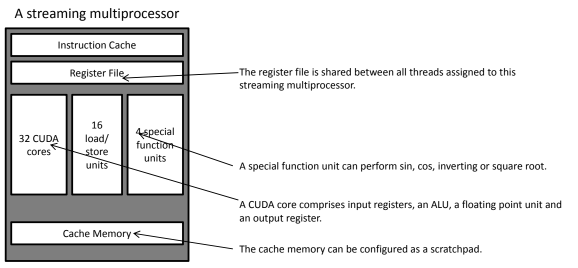
- P1. A method must exist for synchronizing the thread on the main processor and the threads on the GPU and the schedulability analysis on the main processor must take this synchronization mechanism into account. One could either (i) assign sub deadlines to the three different phases (shown in Figure 1(a)) or (ii) let the thread on the main processor suspend when not all threads serving it on the GPU has finished. The former approach transforms the problem to many scheduling problems with constrained-deadline sporadic tasks. For the latter approach, we can use scheduling theory which assumes that tasks can self-suspend [4], [5] for a time which is unknown but is upper bounded. A discussion on different models for describing such suspension in the context of GPU computing is available in [6].
- P2. A method must exist for determining if the GPU should be used to assist a thread on the main processor. This amounts to the task-assignment problem for heterogeneous multiprocessors which is known to be a very challenging problem (it is NP-hard and the standard use of normal bin-packing heuristics, such as first-fit, can cause poor performance [7]). But fortunately, in many practical scenarios, there are only two types of processors available (processor cores of the main processor and the processor cores in the GPU) and for such situations, an efficient algorithm can be created [7].
- P3. A method must exist for determining the finish-


Main processor
GPU
phase-1
phase-2
phase-3
Thread 1 on main processor arrives
deadline of thread
Thread 1 invokes threads on GPU
Thread 1 must wait until Thread 2, Thread 3, ..., Thread k has finished
Thread 2 on GPU arrives.
Thread 2 finishes execution
Thread 3 on GPU arrives.
Thread 3 finishes execution
thread k on GPU arrives
Thread k finishes execution
time

(a) A common use of a GPU. A thread executing on a main processor arrives and after some of its execution, it suspends and invokes multiple threads on the GPU. When these threads have finished, the thread on the main processor resumes execution again.


NVIDIA Fermi GPU
SM_1
SM_2
...
SM_8
Shared Cache
SM_9
SM_{10}
...
SM_{16}
A thread block assigned to SM_1
Another thread block assigned to SM_1
A thread block assigned to SM_2
...

(b) The internals of NVIDIA Fermi GPU. It consists of 16 streaming multiprocessors which share a cache memory. To the right is shown thread blocks; a thread block is assigned to a streaming multiprocessor.


A streaming multiprocessor
Instruction Cache
Register File
32 CUDA cores
16 load/ store units
4 special function units
Cache Memory
The register file is shared between all threads assigned to this streaming multiprocessor.
A special function unit can perform sin, cos, inverting or square root.
A CUDA core comprises input registers, an ALU, a floating point unit and an output register.
The cache memory can be configured as a scratchpad.

(c) A detailed view of a streaming multiprocessor.

Figure 1. The use of a GPU and its internals.

ing time of the group of threads executing on the GPU.

P1 and P2 have partly been addressed in previous research. The current research literature offers no method for P3 however and therefore, we will discuss P3.

Figure 1(b) shows the internals of a GPU; we consider the most recent one – NVIDIA Fermi. It comprises 16 so-called *streaming multiprocessors* (SMs) and a shared cache memory. Software threads are organized into so-called *thread blocks*, where a thread block is assigned to a streaming multiprocessor. A streaming multiprocessor may be assigned many thread blocks but a thread block cannot be assigned to two or more streaming multiprocessors. In order to solve P3 we therefore need to address two subproblems:

P31. Given that the assignment of thread blocks to streaming multiprocessors is known, compute, for each streaming multiprocessor, an upper bound on the finishing time of the threads assigned to this streaming multiprocessor.

P32. Assuming that the exact assignment of thread blocks is not known but some knowledge of the assignment heuristic is available (for example assign thread blocks in a round-robin fashion), compute, for each streaming multiprocessor, an upper bound on the finishing time of the threads assigned to this streaming multiprocessor.

Given that chip makers of GPUs currently do not publish the heuristic used for thread-block assignment, we focus on P31 and postpone P32 for future work. P31 cannot be solved through normal Worst-Case Execution Time (WCET) analysis [8] methods because they assume that all hardware resources of a processor is dedicated to a thread that executes. But this assumption is not true for a streaming multiprocessor. Figure 1(c) shows a detailed view of a streaming multiprocessor. A streaming multiprocessor comprises 32 Compute Unified Device Architecture (CUDA) cores; each CUDA core is composed of an ALU, a floating point unit, input registers and an output register. It is therefore possible for 32 threads to perform ALU operations in parallel on a single streaming multiprocessor. But there are only 16 load/store units; hence only 16 threads can perform load instructions in parallel. Analogously, only a limited number of trigonometrical computations can be performed in parallel. We can see that we need a method for WCET analysis which analyzes not only a single thread which has all hardware resources under its control but instead we must analyze a *set of threads* which *share* hardware resources.

Therefore, in this paper, we present a simple method for calculating an upper bound on the finishing time of threads assigned to a streaming multiprocessor. In order to take this first step, we limit ourselves to the study of a simplified version of a streaming multiprocessor in NVIDIA Fermi.

II. System model

Program Structure: We consider the code structure shown in Figure 2 (derived from matrix multiplication code) for analyzing the finishing time of a set of threads. The '+' mark over the sub-block (involving LOAD, LOAD and a CUDA instruction) in Figure 2 indicates that the sub-block may repeat itself one or more times. The program structure of Figure 2 consists of:

1) an arithmetic instruction (say, initializing a register) that will be executed on a CUDA core followed by
2) a sub-block (which can occur one or more times) consisting of two memory accesses (say, reading two variables into registers) carried out by LOAD/STORE unit and an arithmetic instruction (say fused multiply and add) followed by
3) another memory access instruction (say, to store the result of fused multiply and add instruction).

Assumptions: In order to calculate an upper bound on the finishing time of a *kernel* (a set of thread blocks

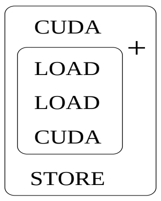
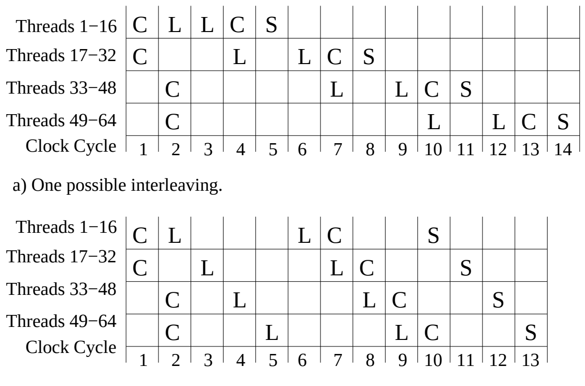


Figure 2. The template of the program's control flow. CUDA means an instruction that uses the CUDA processor (e.g., an ALU operation), LOAD is an instruction that fetches data from memory to a register and STORE is an instruction that stores the content of a register to memory.



Threads 1−16	C	L	L	C	S									
Threads 17−32	C			L		L	C	S						
Threads 33−48		C					L		L	C	S			
Threads 49−64		C								L		L	C	S
Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14

a) One possible interleaving.

Threads 1−16	C	L				L	C			S			
Threads 17−32	C		L				L	C			S		
Threads 33−48		C		L				L	C			S	
Threads 49−64		C			L				L	C			S
Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13

b) Another possible interleaving.

Figure 3. Two possible interleavings/schedules when sub-block has appeared only once.

which serve a thread on the main processor), we make the following assumptions on the SM and its scheduling algorithm:

- Before run-time, tasks assigned to a SM are organized into groups of 16 threads (also known as a *warp*).
- Each instruction takes just one clock cycle to execute (including memory access instructions).
- There are no cache misses i.e., all the data that memory access instructions are interested in is found in cache (it is intuitive from the previous assumption).
- Each SM has 32 CUDA cores and 16 LOAD/STORE units i.e., in a clock cycle, each SM can either perform 32 arithmetic operations or 16 memory operations.
- At run-time, in each clock cycle, the scheduler of SM selects one or two warps for scheduling.
- Whenever there are warps available for execution, the run-time scheduler must select a warp (work-conserving). (Since we assume that each instruction takes just one clock cycle, every warp is available every time as long as its threads have not yet terminated.)
- As already mentioned, we assume that the control flow of the program is as specified by Figure 2; it comprises a CUDA instruction followed by rep_cnt sub-blocks and then a LOAD/STORE instruction. In addition, we assume that instructions are scheduled in a fair manner on sub-block level, that is, when threads compete for resources, an instruction belonging to sub-block k has priority over an instruction belonging to sub-block k+1 – this will be illustrated in Section III. (The assumption on fairness on sub-block level is probably not realistic for GPUs of today but this assumption has the benefit that it simplifies our analysis.)

III. The new method

In this section, we present our method to determine an upper bound on the finishing time of n threads assigned to a single SM. We assume all these threads have the same program structure that is described in Section II. Recall that we make no assumption on the exact scheduling policy in a SM – we assume that it is work-conserving and sub-block level fair. Hence, for a given set of threads, there are many possible schedules (interleavings) that can be generated. To calculate an upper bound on the finishing time of a kernel, it is essential to know: "Given that a certain number of threads are assigned to a particular SM, what order of interleaving of these threads results in a maximum time (measured in clock cycles) to finish their execution". Figure 3 shows two such possible interleavings when 64 threads are assigned to a SM. Figure 3 shows, in each clock cycle, the instruction of different warps that is being executed by CUDA and LOAD/STORE units. We have used **C**, **L** and **S** to represent *CUDA*, *LOAD* and *STORE* instruction respectively. For example, in Figure 3(a), in the first clock cycle, CUDA instruction of thread 1-16 (say, warp1) and 17-32 (say, warp2) are being executed; in the second cycle, LOAD instruction of warp1 and CUDA instruction of threads 33-48 (say, warp3) and 49-64 (say, warp4) are being executed.

In the first case (Figure 3(a)), the scheduler is trying to finish the execution of one warp before switching to another warp but without violating its *work-conserving* property. In this case, the scheduler has scheduled the threads as follows: whenever possible, warp1 is executed, then warp2, then warp3 and when none of these warps can be executed, it executes warp4. In the second case (Figure 3(b)), the scheduler is trying to give a fair share of the resources to each warp by executing one instruction each from every warp and at the same time exploiting the parallelism whenever possible (to preserve the work-conserving property). As we can observe form this example, the first schedule has the worst case finishing time as it takes 14 clock cycles whereas the second schedule takes only 13 clock cycles to finish.

Hence, we conjecture that the worst case interleaving happens for our program structure when the scheduler tries to schedule threads (group of 16 for the architecture under consideration) so as to finish the execution of a group of 16 threads in a particular order (as shown in Figure 3(a)).

Now, considering our conjectured worst possible interleaving for a given number of threads on a SM, we discuss the worst-case finishing time for a given number of threads on a SM. From the schedule shown in Figure 3, we can observe that, for the program structure under consideration, each warp (except the first one) needs 3 additional clock

Threads 1–16	C	L	L	C						L	L	C	S									
Threads 17–32	C			L	L	C						L		L	C	S						
Threads 33–48		C				L	L	C							L		L	C	S			
Threads 49–64		C						L	L	C								L		L	C	S
Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Figure 4. A schedule when sub-block is repeated twice.

Threads 1–16	C	L	L	C						L	L	C						L	L	C	S									
Threads 17–32	C			L	L	C						L	L	C						L		L	C	S						
Threads 33–48		C				L	L	C						L	L	C							L		L	C	S			
Threads 49–64		C						L	L	C						L	L	C								L		L	C	S
Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Figure 5. A schedule when sub-block is repeated thrice.

cycles to finish its execution compared to its previous warp. Note that in Figure 3, the inner sub-block (consisting of LOAD, LOAD and CUDA instructions) appears only once in the program structure. This can be generalized to n threads: The total number of clock cycles needed to finish the execution of n threads when the sub-block appears only once in the program structure is: $\lceil \frac{n}{16} \rceil \cdot 3 + 2$.

Now, consider the same number of threads (i.e., 48 threads) but with the sub-block repeated multiple times. To understand the finishing times in such a scenario, consider schedules for the cases when the sub-block has repeated twice and thrice in Figure 4 and 5. As we can observe, the total number of clock cycles (required to finish the execution of all the 48 threads) for each repetition of sub-block increases by 8. This can be generalized to n threads: The number of additional clock cycles needed to finish the execution of n threads when the sub-block has appeared rep_cnt of times is: $\lceil \frac{n}{16} \rceil \cdot (rep\_cnt - 1) \cdot 2$.

Hence, with the help of above two relations, we conjecture that the total number of clock cycles needed to finish the execution of n_i threads with rep_cnt appearances of the sub-block assigned to SM_i is:

$$FT_i = \left(\left\lceil \frac{n_i}{16} \right\rceil \cdot 3 \right) + 2 + \left\lceil \frac{n_i}{16} \right\rceil \cdot (rep\_cnt - 1) \cdot 2$$

This equation gives the finishing time of n_i threads assigned to SM_i. As described earlier, a GPU consists of many such SMs (say m) and hence the maximum finishing time of a *kernel* is: $FT_{kernel} = \max(FT_1, FT_2, \ldots FT_m)$.

IV. Conclusion and Future work

We presented a method for calculating an upper bound on the finishing time of threads executing on an NVIDIA Fermi GPU. We left the following problems open: (i) proving mathematically the correctness of our stated upper bounds on finishing times, (ii) relaxing the assumption of fairness on sub-block level, (iii) relaxing the hardware assumptions to allow cache misses and longer latency of floating-point operations, (iv) generalizing the method to other control flows and to user-specified number of LOAD/STORE units and CUDA cores and (v) validating the output of our method by comparing it with experimental runs on real hardware.

Acknowledgments

This work was partially supported by ARTISTDesign Network of Excellence on Embedded Systems Design, funded by the European Commission under FP7 with contract number ICT-NoE-214373 and the Portuguese Science and Technology Foundation (Fundação para Ciência e Tecnologia - FCT) and the Luso-American Development Foundation (FLAD).

References

[1] "NVIDIA CUDA Compute Unified Device Architecture Programming Guide, available at http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf."

[2] "NVIDIA's Next Generation CUDATM Compute Architecture: Fermi, http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf."

[3] "http://techreport.com/articles.x/17670."

[4] K. Bletsas, "Worst-case and best-case timing analysis for real-time embedded systems with limited parallelism," Ph.D. dissertation, The University of York, 2007.

[5] P. Gai, L. Abeni, and G. C. Buttazzo, "Multiprocessor DSP scheduling in system-on-a-chip architectures," in *14th Euromicro Conference on Real-Time Systems (ECRTS 2002)*, Vienna, Austria, Jun. 2002, pp. 231–238.

[6] K. Lakshmanan, S. Kato, and R. Rajkumar, "Problems in scheduling self-suspending," in *RTSOPS 2010: 1st International Real-Time Scheduling Open Problems Seminar in conjunction with the 22th Euromicro Intl Conference on Real-Time Systems*, Brussels, Belgium, Jul. 2010.

[7] B. Andersson, G. Raravi, and K. Bletsas, "Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors," CISTER research unit, ISEP/IPP, Polytechnic Institute of Porto, Porto, Portugal, Tech. Rep. HURRAY-TR-100505, May 2010.

[8] A. C. Shaw, "Reasoning about time in higher-level language software," *IEEE Trans. Software Eng.*, vol. 15, no. 7, pp. 875–889, 1989.

Reusable and Partial WCET analysis

Johan Fredriksson
Mälardalen Real-Time Research Centre
Mälardalen University, Västerås, Sweden

***Abstract*—Due to shorter time-to-market and product cycles, development of embedded software has increasingly become a more iterative and agile process. Engineers often make small corrections late in the development process. However, whenever any part of the software is changed a complete WCET analysis is required. Thus, todays WCET analyses are not tuned towards the late stages of the embedded system development process, where often smaller bugs, or minor code improvements, are discovered and fixed. Consequently, always performing a complete system WCET analysis contradicts the type of fine tuning that the engineers often want to do.**

In previous research we have presented a technique for accurate reusable WCET analysis by classifying software inputs with respect to WCET estimates. In this paper we describe an extension to our previous work for lowering the amount of analyses that have to be made at a code change. We propose methods for associating input classes with parts of the source code for allowing WCET analysis to be partially performed with respect to only updated source code.

I. Introduction

WCET analysis provides bounds on the estimated execution time, and the closer these bounds are to the true worst-case, the higher the value of the analysis. Therefore, software designers often want to restrict performance analysis to parts of the software system, called a context, or a mode.

When software systems are exposed to minor code changes the whole system is re-analyzed, and for large complex system, like e.g., avionics, the WCET analyses may take several days. Therefore there is a strong desire to be able to do partial WCET analysis.

Programming embedded systems is a complex task, and often fine tuning in the late stages of development is needed. Quick product cycles leads to that development become more iterative where engineers make smaller corrections, and often late in the development process. However, whenever a smaller part of the code has been modified the whole WCET analysis most often has to be remade. Thus, todays WCET analyses are not tuned towards the late stages of the embedded system development process, where often smaller bugs are discovered and fixed, or minor code improvements are made. Consequently, always doing a complete system WCET analysis contradicts the type of fine tuning that the engineers often want to do.

The overall goal of this work is to provide means for faster estimation of timing behavior in embedded real-time systems. Specifically we propose a method for reducing the amount of analyses that needs to be remade to derive program timing guarantees when only limited parts of the system code have been modified.

II. Related Work

To the best of the authors knowledge, there has been little research on the topic of partial WCET analysis. In this section we describe some previous work in the area WCET analysis that touches upon similar aspects.

Staschulat et al. [1] make a partitioning of execution time behavior of software modules based upon the context in which the module is derived. Our approach have similarities to this work, but we use the partitioning for providing reusable and parametric analysis, whereas Stachulat et al. use partitioning only for increasing the accuracy of WCET analyses. Recent case-studies show that it is important to consider mode- and context-dependent WCET estimates when analyzing real sized industrial software systems [2]. Gheorghita et al. in [3] use usage scenarios to determine tighter loop bounds. In [4] Mohan et al. use run-time usage information for dynamic voltage scaling depending on the timing requirements. Wenzel et al. [5] use both model checking and genetic algorithms to derive which input data that makes a certain instrumented code part to be executed. Gross et al. [6] use evolutionary testing with measurement-based WCET analysis to find a context dependent WCET. In [7] David and Puaut present a framework for probabilistic WCET with static analysis is presented. The probabilities are related to the probability of possible values of external and internal variables. In [8], Bernat et al. analyze each basic block of a program with respect to execution-times, and derived probability distributions of the execution-times. This method is, in contrast to our method, based on measurements. In [9] Lee et al. develop a framework that considers the usage of a system; however, neither software components nor reuse is considered. Ji et al. [10] divide the source code in modes depending on input, and only the parts that are used in a specific usage are analyzed.

III. Reusable WCET analysis

In previous work [11], [12] we have proposed a method for increasing the accuracy of software components WCET by classifying input data with respect to execution-times. We use binary search heuristics to efficiently create parameterizable contracts as a function of a input data to determine the WCET estimate for specific usage scenarios.

The reusable WCET analysis is based on a combination of static WCET analysis and systematic search over the value space of component input variables, for deriving a parameterizable WCET of a reusable software component. We use the input-sensitiveness of computer software components to express a relationship between component inputs and execution times. A WCET contract is defined as a set of input classes with corresponding WCET estimates. The contract is not created with respect to any specific usage, but describes the execution time solely with respect to inputs. A WCET estimate is a context-dependent property that varies depending on inputs, internal software and hardware states.

Consider a software component c_i with a nuber of provided inputs $p_{i,j}$. Each provided input $p_{i,j}$ is associated with a variable $v_{i,j}$ and a value domain $v_{i,j} \in \mathcal{V}_{i,j}$. The set of variables $v_{i,0}, v_{i,1}, ..., v_{i,n-1}$ represent an *input value space* as a set of input value combinations described as tuples $\langle v_{i,0}, v_{i,1}, \ldots, v_{i,n-1} \rangle$.

Definition 3.1: An input variable $v_{i,j}$ represents the value of a provided input $p_{i,j}$, and $\mathcal{V}_{i,j}$ represents the possible values for $v_{i,j} \in \mathcal{V}_{i,j}$.

Definition 3.2: An input value space $\mathbf{D}_i$ is a set of input value combinations, such that $\mathbf{D}_i = \{\langle v_{i,0}, v_{i,1}, ..., v_{i,n-1} \rangle : v_{i,j} \in \mathcal{V}_{i,j}\}$.

Consider the two input variables $v_{i,0}$ and $v_{i,1}$, with the value domains $\mathcal{V}_{i,0} = \{v : 0 \leq v \leq 1\}$ and $\mathcal{V}_{i,1} = \{v : 0 \leq v \leq 1\}$, then the possible input variable combinations are described by the input value space $\mathbf{D}_i = \{\langle 0,0 \rangle, \langle 0,1 \rangle, \langle 1,0 \rangle, \langle 1,1 \rangle\}$.

An input combination tuple $\mathbf{d}$ represents one combination of single values on the provided inputs. Consider a provided interface $P_i = \{p_{i,0}, p_{i,1}\}$ and the input combination $\mathbf{d} = \langle 0, 1 \rangle$ represents that the input $p_{i,0}$ takes the value 0 and $p_{i,1}$ takes the value 1. The input value space $\mathbf{D}_i$ represents all possible value combinations tuples $\mathbf{d}$ for a component c_i. In the input-sensitive WCET analysis, $\mathbf{D}_i$ is partitioned in subsets with respect to a predicate ϕ. For example, the predicate ϕ may restrict the input variables $v_{i,0}, v_{i,1}$ and $v_{i,2}$ such that $\phi = [0 \leq v_{i,0} < 1,\ 0 \leq v_{i,1} < 1,\ 0 \leq v_{i,2} < 2]$. In this case, each tuple $\mathbf{d} \in \mathbf{D}_i$ is restricted to the single value 0 for $v_{i,0}$ and $v_{i,1}$ and to the values 0 and 1 for $v_{i,2}$. A predicate ϕ may also express dependencies between input variables, e.g., $\phi = [1 \leq v_{i,j} < 3,\ v_{i,k} < v_{i,j}]$. $\mathbf{D}_{i|\phi}$ represents the tuples in $\mathbf{D}_i$ that fulfill the predicate given in ϕ. Thus, $\mathbf{D}_{i|\phi}$ is a condition subset of $\mathbf{D}_i$ such that all tuples must fulfills the predicate ϕ.

Definition 3.3: A tuple $\mathbf{d} = \langle v_{i,0}, v_{i,1}, ..., v_{i,n-1} \rangle$ describes one input combination for the provided interface P_i.

Definition 3.4: ϕ is a predicate on an input value space $\mathbf{D}_i$, and $\phi(\mathbf{d})$ notates that the tuple $\mathbf{d}$ fulfills ϕ.

Definition 3.5: An input value space partition $\mathbf{D}_{i|\phi}$ is a condition subset of all tuples $\mathbf{d}$ with respect to a predicate ϕ, such that $\mathbf{D}_{i|\phi} = \{\mathbf{d} \in \mathbf{D}_i : \phi(\mathbf{d})\}$.

Definition 3.6: $|\mathbf{D}_i|$ is the concrete number of input combination tuples in $\mathbf{D}_i$.

We use the concept of input value space partitions with predicates to define a set of input combinations as a use-case. We denote a predicate ϕ^U to define a use-case.

Definition 3.7: A use case $\mathbf{U} = \langle \phi^U, \mathcal{P}, pt \rangle$ is a predicate ϕ^U connected with a probability mass function $\mathcal{P}$ and $0 \leq pt < 1$ is a given priority threshold to ignore low probability WCETs.

A WCET contract is used for obtaining a WCET. The contract is parameterized with the *use case*. All input value space partitions $\mathbf{D}_{i|\phi} \cap \mathbf{D}_{i|\phi^U} \neq \emptyset$ are referred to as *active input value space partitions*, i.e., all input value space partitions $\mathbf{D}_{i|\phi}$ that are *active* and their respective WCETs are eligible for the usage dependent component WCET. A WCET contract is a function of a usage $\mathbf{U}$ that results in a usage dependent WCET. (Equation 1)

$$f_{WCET} : \mathbf{D}_{i|\phi^U} \rightarrow \max_{\forall_i \left(\mathbf{D}_{i|\phi} \cap \mathbf{D}_{i|\phi^U} \neq \emptyset\right)} \left(WCET_{i|\phi}\right) \tag{1}$$

IV. Extension: Partial WCET analysis

The reusable WCET analysis is used to create WCET contracts. Subsets of the software behavior are analyzed by iteratively creating smaller input classes. Assuming that static WCET analysis is used, the software is divided in so called basic blocks *bb*, and each *bb* represents a part of the source code. We also associate an input class with a set of basic blocks, namely those that are analyzed with respect to the input class.

Definition 4.1: a basic block *bb* is a part of a software behavior, and can be related to a source file or an object code file.

Definition 4.2: $bb(\mathbf{d})$ denotes that the basic block *bb* is affected by the input tuple $\mathbf{d}$.

To keep track of all basic blocks that belong to a value space partition, we define a list of basic blocks. The list contains all basic blocks that are affected by the input value space partition.

Definition 4.3: A basic block list $\mathbf{BB}_{i|\phi}$ is a list of all basic blocks *bb* such that $\mathbf{BB}_{i|\phi} = \{bb : bb(\mathbf{d}) \wedge \mathbf{d} \in D_{i|\phi}\}$.

Definition 4.4: $|\mathbf{BB}_i|$ is the concrete number of basic blocks in $\mathbf{BB}_i$.

A basic block contract is used for obtaining all value space partitions that affect a basic block. The contract is used to mark those value space partitions as invalid. Invalid value space partitions need to be re-analyzed.

Definition 4.5: $inv(\mathbf{D}_i)$ denotes that the value space partition $\mathbf{D}_i$ is marked invalid and needs to be re-analyzed with respect to WCET.

A basic block contract is a function of a basic block *bb* and results in the invalidation of a number of value space partitions. The contract is parameterized with a *basic block* derived from a code change. If the basic block *bb* belongs to the list $\mathbf{BB}_{i|\phi}$, then the corresponding value space partition $\mathbf{D}_{i|\phi}$ is considered affected by the code changed and is marked invalid. (Equation 2)

$$f_{BB} : bb \to \underset{\forall_i \left(bb \in \mathbf{BB}_{i|\phi}\right)}{inv(\mathbf{D}_{i|\phi})} \tag{2}$$

The idea is simple - the value space partitions that are invalid are re-analyzed. However, there are a number of challenging problems that we need to handle. For instance, we must ensure that value space partitions that are not marked invalid still have valid WCET estimates. Changed source code may not only affect one basic block, the size of one basic block may also affect where in memory other basic blocks are located. Changing one basic block may in turn lead to that the linker moves whole object files in memory. We try to solve the latter by forcing the linker to layout the object files on specific addresses.

Other problems that we believe that we will encounter are related to cache behavior. When an object file is recompiled and re-linked, small changes caused by internal changes in basic blocks may affect, e.g., the cache behavior.

V. Evaluations

We have only performed a few early simulations on the proposed extension. We have performed the evaluation with two components from an academic ACC example with the components "LoggerOutput" and "SpeedLimit", described in Tables 1 and 2. The ACC example is described in detail in, e.g., [12].

The results indicate that greater number of smaller input value space partitions leads to fewer basic blocks associated with each value space partition. In the best case, each input value space partition is associated with only one basic block. In this way, any source code change always only invalidates one value space partition. Consequently, the wort case is when all basic blocks are associated to all value space partitions.

$\#vsp$	$\|BB_i\|$
1	12
2	8,7
3	5,3,6
4	3,2,3,6

Table I
Number of value space partitions ($\#vsp$) and basic blocks for each value space partition, for "SpeedLimit".

VI. Hardware dependencies

It should be noted that the we only consider input data limits. The execution time is also dependant on the hardware upon which the code is executed and where in memory the code is located. In the case that a simple 4-, 8- or 16-bit CPU is used, which is common in a large segment of the embedded domain, and that the code is forced to reside in and access memory areas with the same timing properties as assumed in the WCET analysis, the WCET estimates derived should also be valid in the new context. However, if a more advanced CPU is used, maybe with a cache or some other performance enhancing features, and the compiler and linker change the code structure, the derived component WCET estimates should be used with caution.

$\#vsp$	$\|BB_i\|$
1	7
2	7,3
3	2,5,3
4	2,2,3,3

Table II
Number of value space partitions ($\#vsp$) and basic blocks for each value space partition, for "LoggerOutput".

VII. Conclusions

In this paper we have presented an on-going extension to our previous work [11], [12], where we have proposed methods for increasing the accuracy of software components WCET by classifying input data with respect to execution-times. The contracts express the relation between input data and execution time. In this way it is possible to determine a parameterized and more accurate WCET by considering the input. In the current on-going research we propose to use input data to express a relation between source code and inputs.

With a good control over the linker, and assigning each object file to a specific address in memory it is possible to recompile and re analyze only affected object files. The advantage of the proposed method is that a smaller part of the code can be re-analyzed, lowering the effort required for re-analyzing the code for small changes.

Further research on hardware dependencies and evaluations are still required.

References

[1] Staschulat, J., Ernst, R., Schulze, A., Wolf, F.: Context sensitive performance analysis of automotive applications. In: Proc. Design, Automation and Test in European Conference and Exhibition (DATE'05), Munich, Germany, IEEE Computer Society Press (2005) 165–170

[2] Sehlberg, D., Ermedahl, A., Gustafsson, J., Lisper, B., Wiegratz, S.: Static wcet analysis of real-time task-oriented code in vehicle control systems. In: Proc. 2^{nd} International Symposium on Leveraging Applications of Formal Methods (ISOLA'06), Paphos, Cyprus, IEEE Computer Society Press (2006) 212–219

[3] Gheorghita, S.V., Stuijk, S., Basten, T., Corporaal, H.: Sharper wcet upper bounds using automatically detected scenarios. Technical Report ESR-2005-04, Eindhoven University of Technology, Department of Electrical Engineering, Electronic Systems (2005)

[4] Mohan, S., Mueller, F., Hawkins, W., Root, M., Healy, C., Whalley, D.: Parascale: Exploiting parametric timing analysis for real-time schedulers and dynamic voltage scaling. In: Proc. 26th IEEE Real-Time Systems Symposium (RTSS'05), Miami, FL, USA, IEEE Computer Society Press (2005) 233–242

[5] Wenzel, I., Rieder, B., Kirner, R., Puschner, P.P.: Automatic timing model generation by CFG partitioning and model checking. In: Proc. Design, Automation and Test in European Conference and Exhibition (DATE'05), Munich, Germany, IEEE Computer Society Press (2005) 606–611

[6] Groß, H.G., Mayer, N.: Evolutionary testing in component-based real-time system construction. In: GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2002) 1393

[7] David, L., Puaut, I.: Static determination of probabilistic execution times. In: Proc. 16th Euromicro Conference of Real-Time Systems, (ECRTS'04), Catania, Sicily, IEEE Computer Society Press (2004) 223–230

[8] Bernat, G., Colin, A., Petters, S.: pWCET, a Tool for Probabilistic WCET Analysis of Real-Time Systems. In: Proc. 3^{d} International Workshop on Worst-Case Execution Time analysis (WCET'03) in conjunction with 13th Euromicro Conference of Real-Time Systems, (ECRTS'03), Porto, Portugal, IEEE Computer Society Press (2003) 21–38

[9] Lee, J.I., Park, S.H., Bang, H.J., Kim, T.H., Cha, S.D.: A hybrid framework of worst-case execution time analysis for real-time embedded system software. In: Proc. Aerospace Conference, Big Sky, MT, USA, IEEE Computer Society Press (2005) 1–10

[10] Ji, M.L., Wang, J., Li, S., Qi, Z.C.: Automated wcet analysis based on program modes. In: Proc. International workshop on Automation of Software Test (AST'06) in conjunction with International Conference on Software Engineering (ICSE'06), Shanghai, China, ACM Press (2006) 36–42

[11] Fredriksson, J.: Improving Predictability and Resource Utilization in Component-Based Embedded Real-Time Systems. PhD thesis, Mlardalen University, School of Innovation Design and Technology, Mlardalen University, Sweden (2008)

[12] Fredriksson, J., Nolte, T., Nolin, M., Schmidt, H.: Contract-based reusable worst-case execution time estimate. In: Proc. of the 13th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07), Daegu, Korea (2007)

Using a Markovian model for branch prediction related WCET

Stéphane Louise, Amira Dkhil
CEA, LIST, MB 94, F91191 Gif-sur-Yvette Cedex, FRANCE
firstname.lastname@cea.fr

***Abstract*—In this paper we use a Markovian model previously applied to cache related delay modeling and preemption delay modeling to a subset of dynamic branch predictors related WCET delays. Our model allows the modeling of a deterministic or non deterministic automaton or set of automata thanks to a Markovian model. We previously applied this formalism to cache related delay and cache related preemption delays [7], [9] and to non deterministic cache policy [8]. Now we would like to introduce an application to some forms of branch prediction associated Worst Case Execution Time (WCET) computation.**

I. Introduction

In this short paper we will focus on the evaluation of a simple local dynamic branch predictor related Worst Case Execution Time (WCET). WCET (and Execution Time as a whole) are of special interest for embedded applications like hard real-time systems or more recently for the compilation process of embedded multicore architectures (for the compilation problem, both Best and Worst CET are required). As a consequence, automatic execution time evaluation should become more and more a field of interest in the near future, as multicore architectures become more pervasive.

As our interest targets are embedded systems, associated embedded processors are usually quite simple, which means that branch predictors, when they are available, are usually also quite simple. A good example are branch predictors based on 2 bits saturating counter like seen in figure 1, which can be encountered *e.g.* in some Freescale PowerPC processors. These branch predictors are simple yet quite efficient especially for short pipelines[6] (usually over 90% of correct prediction over Spec89 benchmarks), and the presence of a branch predictor avoids most waits cycles in the pipeline by speculatively executing instructions. Only when the prediction is wrong the speculative execution is cancelled and the pipeline refilled.

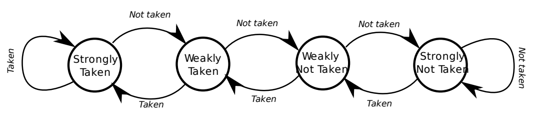


Figure 1. Simple 4 state dynamic branch predictors

We will show in this paper how to use a formalism based on linear algebra previously used on cache related WCET [7], [8] in order to predict also dynamic branch prediction associated WCET overheads. This is a first evaluation of a theorical development. The paper is organized as follows: section II presents the bases of the formalism and the model, section III presents first results on a limited set of usual WCET benchmarks and discuss the results, before concluding and presenting future directions of work in section IV.

II. Formalism

A. Bases

The formalism relies on static analysis, but the static analysis by itself of the Control Flow Graph (CFG) is out of the scope of the paper (an extensive literature exists about it see *e.g.* [10], [5], [3]). Hence we will only focus on the analysis of branch predictor states. As long as static analysis is concerned, there can be 3 possible outcomes of a CFG branch static analysis for each conditional branch:

- Branch taken: the given branch is taken for sure (*e.g.* the $n-1$ increments of a *for* loop with n increments),
- Branch not taken: the given branch is not taken, for sure (*e.g.* the last increment of a simple loop),
- Branch unknown: the given branch can not be statically predicted (*e.g.* because it depends on run time data).

Therefore, for static analysis, the branch predictor state for each conditional branch can be updated by adding several "*unknown*" states, as seen in figure 2.

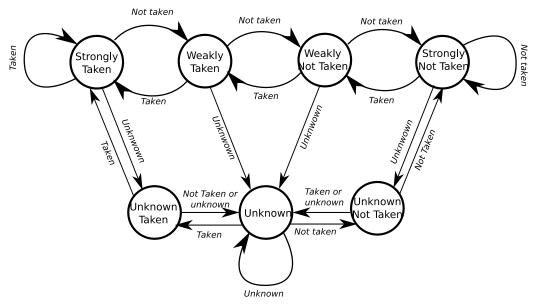


Figure 2. Simple 4 state dynamic branch predictors

The meaning of these unknown states reflects the principle of static analysis where the real state is mapped to an abstract space where properties of interest can be computed (see Cousot & Cousot [4]). The new states, thus, translate the fact that some information can not be statically known about the branch predictor state, typically because of dependence on external data, only available at runtime (control flow hazards).

B. Vector Space

As the static analysis automaton in figure 2 has 7 states, that means that the basic state vector for a given branch instruction state will have also 7 linearly independent states. For a complete state of execution of a given branch prediction, tree additional states must be provided: the number of rightly predicted branches (p), the number of badly predicted branches ($\bar{p}$) and the number of unknown predictions (p_u), meaning that $p + \bar{p} + p_u$ is the total number of execution of the given branch instruction.

The 10 coordinates of the vector space for the branch instruction are:

- p number of rightly predicted branches,
- $\bar{p}$ number of badly predicted branches,
- p_u number of statically unknown predictions,
- T predicted *Taken* state,
- t predicted *weakly taken* state,
- n predicted *weakly not taken* state,
- N predicted *Not Taken* state,
- u_T predicted *unknown taken* state,
- U predicted *Unknown* state,
- u_N predicted *unknown not taken* state.

And the associated state will be a vector

$$s =^t \begin{pmatrix} p & \bar{p} & p_u & T & t & n & N & u_T & U & u_N \end{pmatrix}$$

The global state will be the Cartesian product of all the state spaces associated with each conditional branch instruction of the program: if $s_0, s_1, \ldots, s_{n-1}$ are n conditional branches of a given program, the associated global state is:

$$S = \otimes_{k=0}^{n-1} s_k$$

C. Operators

Operators, in our formalism, modelize the evolution of the state vectors as an instruction is executed. It means that the state after execution, let us note it s_{i+1}, can be derived from the previous state s_i by the application of a given operator W so that $s_{i+1} = W s_i$. This, by definition, describes a Markovian evolution of the vector space. Here, only the conditional branch instructions are modelized.

For static analysis of state updates, we will use 3 operators corresponding to the 3 possible outcomes of a conditional branch as seen previously (*taken*, *not taken*, and *unknown*).

$$W_t = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$W_n = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$W_u = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

It can be noticed that the lower-right side 7×7 matrix of each operator is the transition matrix of the state machine of figure 2 (it is a transfer sub-matrix from one state to the other). The upper 3 lines update the number of right or bad or unknown predictions. For example, in the W_u operator most of the third line is filled with 1 translating that $p_u^{i+1} = p_u^i + 1$ (*i.e.* the number of predictions statically classified as unknown is incremented by 1 after applying W_u onto the branch predictor state).

D. From branch predictor state to Execution Time

We will use simple hypotheses here, as it is only a preliminary work: Branch Prediction penalty is only paid in case of a bad prediction and consist in a complete pipeline invalidation. As we target embedded processors, we will make the assumption that the execution is either *in order* or only slightly *out of order*, so that only the pipeline refilling penalty is paid in this case. Filling the Branch Target Buffer and initializing the Branch History Table (BHT) for the first time a given branch is encountered is paid only once at the first occurrence, therefore once for each branch (hypothesis of perfect cache, which would work well on the tiny examples that we use for our first validation: we will talk in the conclusion about how we can go further here).

Based on these hypotheses, the branch prediction overhead can be calculated as:

$$o_{best} = \sum_{k=0}^{n-1} \bar{p}_k . c_r + c_f . n$$

and

$$o_{worst} = \sum_{k=0}^{n-1} \left(\bar{p} + p_u\right)_k . c_r + c_f . n$$

Where c_r is the cost of pipeline refilling, and c_f the cost of filling the BTB and initializing the predictor state.

III. First evaluations

A. Experimental setup

We compiled the benchmarks using gcc with "-O2" optimization level, targeting MIPS R3000 architecture and their Control Flow Graphs (CFGs) were extracted as input to our model. We have chosen Mälardalen WCET research group benchmark programs, used to evaluate and compare different types of WCET analysis tools and methods [11]. The set of benchmarks is listed in Table I. Our choice included applications with small code size (*e.g.*, sqrt, matmult, bs) as well as medium code size (*e.g.*, crc). Also, our chosen benchmarks contain both single-path programs (*e.g.*, matmult) and multiple-path programs (*e.g.*, crc) with different degrees of complexity in the analysis of the assembly code and the determination of the associated predictor.

B. Analysis methodology

First we analyze the assembler code to extract the Control Flow Graph and also detect the conditional branches. This analysis is very useful to determine whether it is a "*Backward*" or a "*Forward*" branch. Such information will enable us to set the initial state vectors. We have performed these experiments for all of the benchmarks but as an example, we choose the matrix multiplication presented in Figure 3. Three conditional branches were found, which will be the number of the state vectors.

The corresponding state vectors in the case backward is

$$s^b_{0,i} =^t \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where $i = 1\ldots$, and n is the number of conditional branches. Which means that backward branch are preset as *weakly taken* and forward jump as *weakly not taken*.

The origin of such a hypothesis is the well known heuristic BTFN (Backward Taken Forward Not Taken), which differentiate the conditional branches with their way of evolution [1]. s_{i+1} can be derived from the previous state s_i by the application of the appropriate operator W_t, W_n or W_u. The employment of such operators will depend on the analysis results of the CFG, it means that it depend on whether the state of the conditional branch is related to unknown data, for example, or not.

C. Results

The analysis results are shown in Table II. We simulated the branch predictors to get the total number of badly predicted branches.

These results were not difficult to obtain straight from the model. To compute branch predictor related time overheads (both best case and worst case), we used the following figures: $c_r = 5$ cycles and $c_f = 2$ cycles (estimated worst case). The results are shown on figure 3.

Table II
Results of the branch predictor

Benchmark	p	$\bar{p}$	p_u
matmult	7239	381	0
bs	5	1	15
sqrt	17	1	34
crc	18	1	40

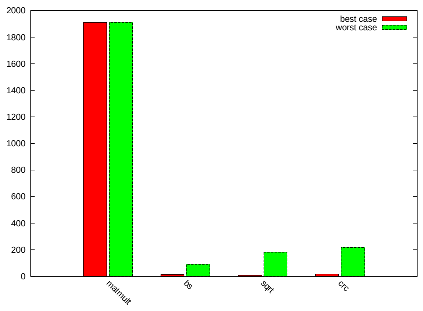


Figure 3. Best and worst case prediction overhead predicted by the model (real best and worst case are the same in the model as the real one)

Of course we need to compute the total branch overhead by taking into account both the several control flow outcomes in conjunction with the evaluation of the predictions. This is shown in figure 4. As expected, *matmult* whose CFG has no hazards is entirely dominated by prediction overhead (that means that without any branch prediction, the branch overhead would be 2 orders of magnitude larger). For the others, with actual CF hazards, the prediction overhead stands for $\approx 10\%$ of the total control flow execution time overhead.

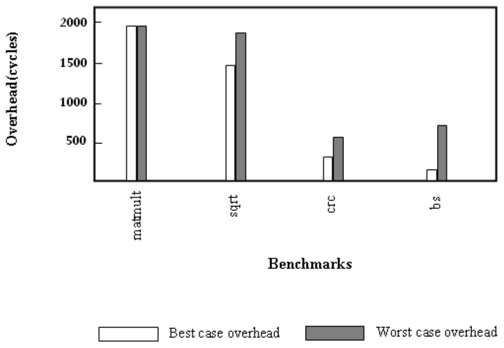


Figure 4. Best and worst case total branch overhead predicted by the model (real best and worst case are the same in the model as the real one)

D. Comments

We compared the results of our first evaluation with the actual real best and worst case overheads, and as expected on quite simple benchmarks, the results of the model match the real worst and best cases. It is also expected that this

Table I
BENCHMARKS DESCRIPTION

Benchmark	Description	Comments	Lines of code
matmult	Matrix multiplication of two 20x20 matrices	Multiple calls to the same function, nested function calls, triple-nested loops	163
bs	Binary search for the array of 15 integer elements	Completely structured	114
sqrt	Square root function implemented by Taylor series	Simple numerical calculation	77
crc	Cyclic redundancy check computation on 40 bytes of data	Complex loops, lots of decisions, loop bounds depend on function arguments, function that executes differently the first time it is called	128

will remain true for a large number or even all the programs as this type of dynamic branch predictor is very simple.

Nonetheless, we made several assumptions here. Firstly, we supposed that the CFG of the program was correctly analyzed. This hypothesis can be endangered by complex CFGs especially when loops exit conditions are tricky (*e.g.* non decidable behaviors). But anyway, good practices of embedded system programming mostly forbid hard or impossible to analyze CFGs, so that best and worst bounds can always be obtained (at least approximate ones).

Secondly, we also made the assumption that the BHT and BTB are infinite (*i.e.* no conflicts can be observed). This is true for these programs since there is typically less than a dozen branches. For larger programs this would not be the case. Nonetheless, our model is based on a Markovian formulation. That means that it is able to handle non deterministic behaviors like cache replacement in control flow hazards (this feature was used to determine cache related WCET in case of preemption in previous works or "exact" WCET associated with non deterministic cache replacement policies). We believe that this can be an asset of our method to characterize branch prediction related execution time overhead in comparison to other methods. Of course, works like [3] or [2] already take into account the finiteness of BTH and BTB and we will need to compare that also. It can also be remarked that most of the time embedded computations have an important spacial locality, even more so for branches than for data accesses, so with reasonably sized BTB, real conflicts are expected to be low on most of applications.

Of course more work is required to strengthen our results but the first benchmarks show that the method is worthy of further investigations.

IV. CONCLUSION AND FUTURE WORK

As a preliminary work, we demonstrated that applying a formalism that we introduced previously for cache related WCET to a simple case of dynamic branch predictor works well on simple benchmarks. Further work is still required to go beyond these first cases, but there is no forecasted issues.

Therefore, our future work will focus on taking into account the finiteness of BHT and BTB. Once the results are consolidated with this first type of dynamic branch predictor, we aim at doing further work with more elaborate branch predictors, as, *e.g.* history based branch predictors which are typically hard to statically take into account and for which a Markovian model would be a real asset.

REFERENCES

[1] Claire Burguière. *Modélisation de la prédiction de branchement pour le calcul de temps d'exécution pire-cas*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France, juin 2008.

[2] Claire Burguière and Christine Rochange. A contribution to branch prediction modeling in WCET analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'05)*, volume 1, pages 612–617, march 2005.

[3] Antoine Colin and Isabelle Puaut. Worst case execution time analysis for a processor with branch prediction. *Real–Time Systems*, 18:249–274, may 2000.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[5] Christopher Healy, Mikael Sjödin, Viresh Rustagi, David Whalley, and Robert Van Engelen. Supporting timing analysis by automatic bounding of loop iteration. *Real–Time Systems*, 18:129–156, may 2000.

[6] J.L.A. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, 4th edition, 2007.

[7] Stéphane Louise. *WCET safe upper bound computation for hard real–time tasks, for systems using cache memories*. PhD thesis, Université Paris XI, UFR scientique d'Orsay, 2002.

[8] Stéphane Louise. A preliminary study of wcet for a non deterministic cache policy. In *Proceedings Work in Progress Session of the 21st ECRTS*, pages 17–20, 2009.

[9] Stéphane Louise, Vincent David, and Jean Delcoigne. A new paradigm for cache related wcet computation. In *NPDPA'02*, 2002.

[10] Thomas Lundqvist and Per Stenström. An integrated path and timing analysis method based on cycle–level symbolic execution. *Real–Time Systems*, 17:183–207, nov 1999.

[11] Mälardalen WCET research group. Wcet benchmarks, http://www.mrtc.mdh.se/projects/wcet/benchmarks.htm.

Divisible Load Scheduling of Real-time Task on Heterogeneous Clusters with Worker Selection Strategy

Suriayati Chuprat and Zuraini Ismail
Universiti Teknologi Malaysia
International Campus, Kuala Lumpur, Malaysia
suria@ic.utm.my, zurainisma@ic.utm.my

***Abstract*— Real-time Divisible Load Theory (RT-DLT) holds enormous potential for modeling an emergent class of parallel real-time workloads. However, the theory needs strong formal foundations before it can be widely used for the design and analysis of real-time systems. As part of our on-going research effort to develop such formal foundations, we recently [11] extended an algorithm based on Linear Programming to schedule a divisible real-time workload upon heterogeneous clusters. Through series of simulation, we observed that the features of clusters have a significant impact upon the completion time of a workload execution. In this paper, we proposed an enhancement to the framework for scheduling a divisible real-time workload upon heterogeneous clusters.**

Keywords-Divisible Load Scheduling, Parallel, Real-time Systems, Heterogeneous Clusters

I. INTRODUCTION

Real-time computer application systems are systems in which the correctness of a computation depends upon both the logical and temporal properties of the result of the computation. As the functionality demanded of such real-time application systems has increased, these systems are increasingly coming to be implemented upon *multiprocessor* platforms. However, the formal models for representing real-time workloads have traditionally been designed for the modeling of processes that are expected to execute in *uniprocessor* environments. These traditional models fail to capture some important characteristics of multiprocessor real-time systems and imposed additional restrictions. One particular such restriction is that each task may execute upon at most one processor at each instant in time. We believe that this is overly restrictive for many current multiprocessor platforms and in fact one significant causal factor of much of the complexity of multiprocessor scheduling [1]. Moreover, the next generation of embedded and real-time systems undoubtedly will demand parallel executions. Looking at these significant needs, recently, some researchers [9, 10, 14, 15, 16, 17] have studied extension to the traditional workload models, to allow for the possibility that a single job may execute simultaneously on multiple processors.

In a series of papers [4, 5, 6, 7, 8], Lin et al. have applied divisible load theory (DLT) [2, 3] to real-time workloads. In DLT, such workloads, it is pointed out in [4, 5, 6], are quite common in data-intensive applications from domains as diverse as bioinformatics and high energy particle physics (e.g., the Compact Muon Solenoid and the ATLAS [AToroidal LHC ApparatuS] projects associated with the Large Hadron Collider at CERN. Lin et al. extended DLT to apply to divisible real-time jobs, that is divisible jobs with associated deadlines and the requirement that a job complete by its deadline in order to be useful. We refer this promising workload model as RT-DLT.

We improved upon *approximation* algorithm as in [7,8] by providing *exact* efficient algorithms [12,13] to determine the smallest number of processors needed to complete the divisible job by its deadline and minimize the completion time. One assumption made in the previous work on RT-DLT is that all processors are homogeneous, in sense that each processing node has the same computational power. In our most recent work [11], we have extended the algorithm proposed in [12, 13] to schedule a real-time divisible workload upon heterogeneous processors and observed that the features of cluster have a great impact in minimizing the completion time of a workload and complete by its deadline. Thus, particularly in heterogeneous clusters, a worker selector needs to be introduced.

The remainder of this paper is organized as follows. In Section 2, we briefly describe the foundations of RT-DLT. The proposed scheduling framework and preliminary simulation work will be presented in Section 3. Finally we conclude and propose some future research agenda in Section 4.

II. RT-DLT: FOUNDATIONS

We now describe the foundations of RT-DLT used in this research. We keep our discussion brief; please refer to [4, 5, 6] for the motivation for this model, and for important emerging example applications that are accurately and conveniently modeled in it.

Job Model. The job model in RT-DLT allows for the simultaneous execution of a job upon multiple processors. Each divisible job J_i is characterized by a 3-tuple (A_i, σ_i, D_i)**,** where $A_i \geq 0$ is the arrival time of the job, $\sigma_i > 0$ is the total load size of the job, and $D_i > 0$ is its relative deadline, indicating that it must complete execution by time-instant $A_i + D_i$.

System Model. The computing cluster used in DLT is comprised of a head node denoted *P₀*, which is connected via a switch to *N* processing nodes denoted P_1, P_2...P_N. Each processing node has the same computational power, and all the links from the head to the processing nodes have the same bandwidth. It is assumed in [4, 5, 6] that:

- The head node does not participate in the computation – its role is to accept or reject incoming jobs, execute the scheduling algorithm, divide the workload and distribute data chunks to the processing nodes.
- Data transmission does not occur in parallel. However, computation in different processing nodes may proceed in parallel to each other.
- The head node, and each processing node, is non-preemptive: the head node completes the dividing and distribution of one job's workload before considering the next job, and each processing node completes executing one job's chunk before moving on to the chunk of any other job that may have been assigned to it.
- Different jobs are assumed to be independent of one another.

In [4, 5, 6], linear models are used to represent transmission and processing times. The computation time of a load of size σ is equal to $\sigma \times C_m$, while the processing time is equal to $\sigma \times C_p$, where C_m is a cost function for transmitting a unit workload and C_p is a cost function for processing a unit workload. For the kinds of applications considered in [4, 5, 6], the output data is just a short message and is assumed to take negligible time to communicate. For a given computing cluster, $\beta \equiv C_p / (C_p + C_m)$.

Figure 1 depicts an example of a time diagram of a RT-DLT scheduling. For a given job (A, σ, D) and a given number of processing nodes *n*, let $\sigma \times \alpha_j$ denote the amount of the load of the job that is assigned to the *j*'th processing node, $1 \leq j \leq n$.

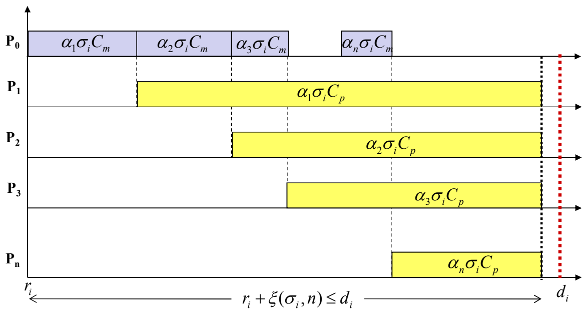


Figure 1. Time diagram of a RT-DLT scheduling

Since data-transmission occurs sequentially, the *i*'th node P_i can only receive data after the previous $(i-1)$ nodes have completed receiving their data. Hence, each P_i receives its data over the interval: $\left[C_m \sum_{j=1}^{i-1} \alpha_j \sigma, C_m \sum_{j=1}^{i} \alpha_j \sigma\right)$ and therefore completes execution at time-instant: $C_m \sum_{j=1}^{i} \alpha_j \sigma + C_p \alpha_i \sigma$. Letting $\xi(\sigma, n)$ denote the time-instant at which the job completes execution, and observing that this completion time is given by the sum of the data-transmission and processing times on P_i, we have:

$$\xi(\sigma, n) = \alpha_1 C_m \sigma + \alpha_1 C_p \sigma$$

$$\equiv \xi(\sigma, n) = \frac{1-\beta}{1-\beta^n} \sigma (C_m + C_p) \qquad (1)$$

Scheduling Framework. Figure 2 depicts an abstraction of the scheduling framework as proposed by Lin et al. [4, 5, 6] combining scheduling algorithms, node assignment strategies, and task partitioning strategies. Each arrival jobs $J_i = (A_i, \sigma_i, D_i)$ will be placed in the *Jobs Queue*. The *Scheduling Strategy* decide the execution order, the *Node Assignment Strategy* compute the number of processors needed for a task execution and *Partitioning Strategy* distribute a job into subtasks and send each subtask to each processors via a switch.

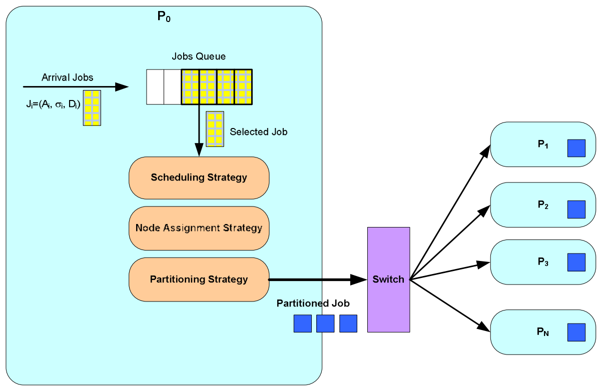


Figure 2. The abstraction of RT-DLT framework

Min- ξ

subject to the following constraints:

(1) $\alpha_1+\alpha_2+\alpha_3+\alpha_4=1$

$$(2)\quad \begin{aligned} r_1 &\le s_1\\ r_2 &\le s_2\\ r_3 &\le s_3\\ r_4 &\le s_4 \end{aligned}$$

$$(3)\quad \begin{aligned} s_2 &\ge s_1+\alpha_1\sigma C_{m1}\\ s_3 &\ge s_2+\alpha_2\sigma C_{m2}\\ s_4 &\ge s_3+\alpha_3\sigma C_{m3} \end{aligned}$$

$$(4)\quad \begin{aligned} s_1+\alpha_1\sigma(C_{m1}+C_{p1}) &\le \xi\\ s_2+\alpha_2\sigma(C_{m2}+C_{p2}) &\le \xi\\ s_3+\alpha_3\sigma(C_{m3}+C_{p4}) &\le \xi\\ s_4+\alpha_4\sigma(C_{m4}+C_{p4}) &\le \xi \end{aligned}$$

Figure 3. Computing the completion time – Modified ***MIN-ξ*** for heterogeneous clusters, N=4

III. EXTENDING *MIN-ξ* AND FRAMEWORK MODIFICATION

In [11] we recently customized the formulated LP, ***MIN-ξ*** [12,13] to compute the earliest completion time upon more general heterogeneous platforms in which there may be a different C_m and a different C_p associated with the data-communication and computing capacity of each processor P_i. Figure 3 depicts the heterogeneous transformation for cluster N=4.

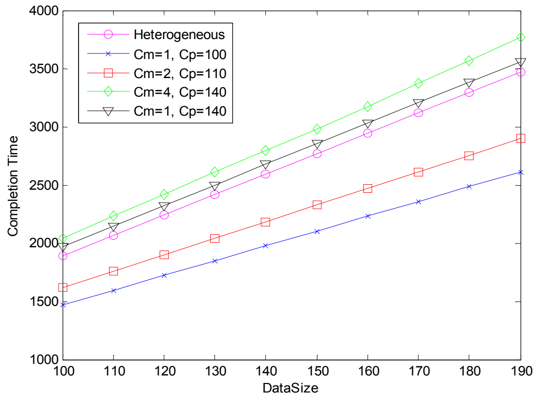


Figure 4. Simulation result – Modified ***MIN-ξ*** for heterogeneous clusters, N=4

Figure 4 demonstrates the impact of different values of C_m and C_p in the heterogeneous clusters. As obviously observed, the completion time increases as C_m or C_p increases. As shown in Figure 5, we proposed a modification to the existing RT-DLT scheduling framework – that is to include a module called *Worker Selection Strategy*. Among other strategies that considered in this module are:

- The scheduler should select the suitable communication links – Minimum communication cost C_m would minimize the completion time, thus should be assigned to jobs with critical deadlines.
- Similarly, the scheduler should select the most appropriate processors – Minimum computation cost C_p would further minimize the completion time should be also assigned to jobs with critical deadlines.
- The scheduler should also assign more processors to a job with critical deadlines.

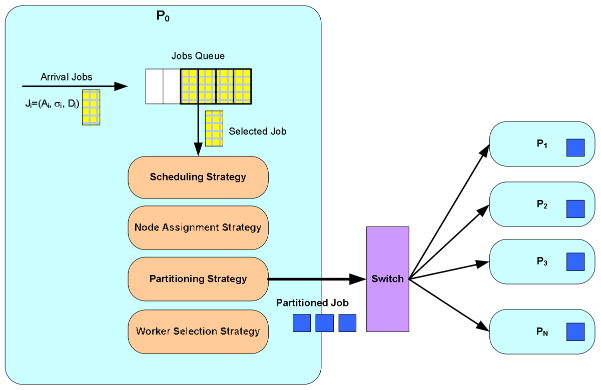


Figure 5. Modified Scheduling Framework

This phase of the research is ongoing; we are currently building the simulation model to evaluate the modified scheduling framework. The second step, which is currently in a very preliminary phase, is to draw convincing hypotheses based on the outcomes of the simulation results, and to use current real-time scheduling theory to explain these hypotheses (extending the theory if the need arises).

IV. CONCLUSION

Most workload models currently used in real-time scheduling theory restrict each job to execute upon at most one processor at each instant in time. This restriction is increasingly proving to be unrealistic upon modern multiprocessor platforms; hence, new models are needed. One of the more promising approaches in this respect has been the recent work of Lin et al. that applies divisible load theory (DLT) to multiprocessor real-time systems. We believe that DLT and its variants hold tremendous promise as potential sources of ideas for providing such general models. In recent work [11, 12, 13], we have studied and extended the work on RT-DLT initiated by Lin et al. in [4, 5, 6]. We are currently studying further generalizations that more accurately reflect the reality of actual multi-cluster computing environments and building theoretical foundations.

REFERENCES

[1] C. L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. JPL Space Programs Summary 37-60, II:28-31, 1969.

[2] V.Bharadwaj, D.Ghose, V.Mani and T.G.Robertazzi. Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamitos CA, September, 1996.

[3] T. G.Robertazzi. Ten reasons to use divisible load theory. Computer, 36(5):63–68, 2003.

[4] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. Technical Report UNL-CSE-2006-0016, Department of Computer Science and Engineering, The University of Nebraska at Lincoln, 2006.

[5] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for clusters. In Proceedings of the Real-Time Systems Symposium--Work-In-Progress Session, pp. 9-12, Rio de Janerio, December 2006.

[6] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. In Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS), pp. 303-314, Bellevue, Washington, April, 2007.

[7] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling with different processor available times. In Proceedings of The International Conference on Parallel Processing (ICPP), Xian, China, September 2007.

[8] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Enhanced real-time divisible load scheduling with different processor available times. In Proceedings of the 14th IEEE International Conference on High Performance Computing (HiPC), pp. 308-319, Goa, India, December 2007.

[9] X. Lin, J. Deogun, Y. Lu and S. Goddard. Multi-round Real-Time Divisible Load Scheduling for Clusters. In Proceedings of 15th International Conference on High Performance Computing (HiPC), pp. 196-207, Bangalore, India, December 2008.

[10] A. Mamat, Y. Lu, J. Deogun and S. Goddard. Real-Time Divisible Load Scheduling with Advance Reservation. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), pp. 37-46, Prague, Czech Republic, July 2008.

[11] S. Chuprat. Divisible Load Scheduling of Real-time Task on Heterogeneous Clusters. In Proceedings of the International Symposium on Information Technology (ITSim 2010, co-sponsored by the IEEE), Kuala Lumpur, Malaysia. June 2010.

[12] S. Chuprat, S. Salleh and S. Baruah. Evaluation of a linear programming approach towards scheduling divisible real-time loads. In Proceedings of the International Symposium on Information Technology (ITSim, co-sponsored by the IEEE), I:455-462, Kuala Lumpur, Malaysia. August 2008.

[13] S. Chuprat and S. Baruah. Scheduling Divisible Real-Time Loads on Clusters with Varying Processor Start Times. In Proceedings of the IEEE 14th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 15-24, Kaohsiung, Taiwan. August 2008.

[14] G.Manimaran and C. S. R. Murthy. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. IEEE Transaction on Parallel and Distributed Systems, 9(3):312–319, 1998.

[15] W. Lee, S. J. Hong, and J. Kim. On-line scheduling of scalable real-time tasks on multiprocessor systems. Journal of Parallel and Distributed Computing, 63(12):1315-1324, 2003.

[16] S. Collette, L. Cucu, and J. Goossens. Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited parallelism. In Proceedings of the 15th International Conference on Real-Time Systems (RTNS), pp. 123-128, Nancy, France, March 2007.

[17] S.Collette, L.Cucu, and J.Goossens. Integrating Job Parallelism in Real-Time Scheduling Theory. Information Processing Letters, 106(5):180-187, May, 2008.

Temperature-Aware Online Real-Time Scheduling for Multiple Feasible Intervals

Bo Liu, Fang Liu, Jian Lin, Albert M.K. Cheng
Department of Computer Science
University of Houston
Houston, TX 77204, USA
e-mail: {boliu, fliu, jlin6, cheng}@cs.uh.edu

Stefan Andrei
Department of Computer Science
Lamar University
Beaumont, Texas 77710, USA
e-mail: sandrei@cs.lamar.edu

Abstract—**In this paper, we propose a temperature-aware online real-time scheduling algorithm for Multiple Feasible Intervals task model which aims to reduce the power consumption subject to the operation temperature threshold for low-power Multiprocessor System-on-Chip platforms. To the best of our knowledge, we are the first to address this problem with details of our algorithm.**

Keywords-embedded system; temperature-aware; real-time scheduling; Multiple Feasible Intervals; low-power; Multiprocessor system-on-chip(MPSoC)

I. Introduction

Multiple Feasible Intervals (MFI) is a new type of aperiodic task model that was first addressed in [7, 8]. In MFI task model, each task can have multiple feasible intervals for execution. Once a task chooses one of its intervals to execute, it needs to complete its work within this interval. Otherwise, it has to choose another interval to start over. Most of time, the intervals may not be known beforehand and the intervals may be changed during run-time, e.g. when new tasks arrive or network congestions occur. We call a MFI task model with changing feasible intervals Dynamic Multiple Feasible Intervals (DMFI) task model.

As the system power consumption soars in modern microprocessors, we proposed a few power-aware MFI scheduling schema including both online algorithms for DMFI [9, 26] and offline algorithms for MFI [10]. While we target on reducing the power consumption and lowering deadline missing ratios for MFI tasks, we did not take thermal issues into consideration. Because the power density directly transforms into heat, the temperature in modern integrated circuits increases dramatically due to smaller feature size e.g. recently announced 28nm technology, higher packing density, and rising power consumption [2, 16], it is critical to tackle thermal issues in all levels of the system design.

A wide range of thermal-issue studies for integrated circuit (IC) design has been delivered in a decade [12, 13, 14, 17, 19, 20, 21, 22]. Temperature affects not only the power but also the performance, reliability, and cost of embedded systems. Specifically, three problems arise due to the high rising operation temperature. First, the higher the temperature becomes, the more leakage power is consumed. The leakage power increases exponentially with the temperature increase, which in turn causes further temperature increase and might incur the well-known thermal runaway problem. Second, a higher temperature also reduces the system's lifetime. Third, a higher temperature impacts the performance of the transistors and usually decreases the performance of circuits. As a consequence, deadline missing ratios become higher.

Studies of thermal energy by Paci et al. [14] indicate that power dissipation is negligible for low-power platforms with their experiments on the feature size as small as 65nm. However, researchers recently start to address temperature-aware scheduling on low-power embedded systems due to the trend of adopting Multiprocessor System-on-Chip (MPSoC) into their designs. Two categories of research are published. One is on the architecture level during design time [2, 6, 16, 17, 18, 19]. It either targets on solving thermal problems from complete hardware point of view, or on solving them with Hardware/Software co-synthesis which is a combination of the two. The other is on the application level [1, 4, 11, 15]. It aims to solve thermal problems for general platforms such as smart phones and future personal mobile devices, which have various hot-spot profiles of applications on these platforms. Thus, application-oriented scheduling with design flexibility works better on them.

In this paper, we introduce a multilevel temperature-aware online DMFI scheduling on the application level. It extends Online Dynamic Multiple Feasible Intervals (ODMFI) introduced in [26]. It scales CPU frequency to bring down energy consumption subject to the operation temperature threshold. Since frequency and temperature have impacts on each other, the task allocation and the workload on each core are adjusted dynamically based on combined profile of applications and the platform. Thus, the operation frequency is scaled up and down based on available slacks, the estimated execution time built from *priori*, and the temperature threshold. To the best of our knowledge, the thermal problem for Dynamic MFI task sets [10] was not yet addressed.

Dynamic scaling techniques, including Dynamic Voltage Scaling (DVS) and Dynamic Voltage Frequency Scaling (DVFS), are commonly used in temperature-aware scheduling, for instance [1, 4, 11, 15, 17]. However, none of them targets on the same system model as ours. Specifically, Bao, Andrei, Eles and Peng in [17] propose voltage selection enabled by DVS on architecture level; Wang and Bettati in [1] extend general task arrivals with First-in First-out scheduling and Static Priority scheduling; Chantem, Dick and Hu in [4] minimize peak temperature subject to time constraints with static task allocation; and Chen, Wang and Thiele in [15] propose a temperature-aware scheduling extending Earliest Deadline First (EDF) scheduling. The closest work we found is done by Fisher, Chen, Wang and Thiele in [11] where they describe a two-phase thermal-aware global scheduling: an ideally preferred speed is calculated in the first phase to minimize peak temperature for the tasks, and a feasible speed is further calculated by increasing the preferred speed in the second phase in order to guarantee global scheduability.

The rest of this paper is organized as the following. Section II provides background of thermal issues on MPSoC platforms and derives our temperature function. Section III proposes the multilevel temperature-aware ODMFI (MT-ODMFI) which has a task allocator distributing work to safely operating cores from temperature point of view, and a local scheduler who executes each task contingent on both timing and temperature restrictions. Section IV describes some experiment design issues, and Section V outlines our future work.

II. Thermal Model for Low-Power MPSoCs

There are two kinds of power consumption on ICs: dynamic power consumption and leakage power consumption [2, 3, 4, 12, 13, 14, 15]. Dynamic power or switching power is the result of charge and subsequent discharge of digital circuits. Leakage power or static power is the result of leakage currents in CMOS circuits. The dynamic power consumption can be calculated using (1):

$$P_{dynamic} = C \cdot V^2 \cdot f. \tag{1}$$

where C is the collective switching capacitance, V is the power supply voltage and f is the operation frequency.

As mentioned earlier, the leakage power consumption grows exponentially, and the dominant types will be subthreshold leakage and gate leakage [4, 13, 15]. For low-power MPSoCs (LP-MPSoCs), leakage power can be approximated by a linear function in the operating temperature ranges of integrated circuits with roughly 5% error [3, 14]. Thus, we use the leakage power function defined in (2):

$$P_{leakage} = \alpha \cdot \theta(\tau) + \beta. \tag{2}$$

where α, β are constants and θ(τ) is a function of time τ for the operation temperature.

Thermal models derived from (1) and (2) have different forms in a wide range of studies. One of the most influential studies is about the relationship between a function unit's temperature and its frequency formalized as a super-linear function in [1]. Since we use the temperature model in online scheduling algorithm, we follow the *simple* rule and choose a simplified temperature function defined in (3):

$$\Theta(\tau) = (\Theta(\tau_0) - a \cdot f_C^{\alpha}/b) \cdot e^{-b(\tau-\tau 0)} + a \cdot f_C^{\alpha}/b. \tag{3}$$

where a, b, α are curve fitting constants, and $\Theta(\tau_0)$ is the operation temperature at any start point τ_0. Since we assume no other dynamic workload except MFI task sets, τ_0 can be any time point when system is idle or in an *equilibrium* state. The system is in the *equilibrium* state before all the tasks arrive, and we can derive $\Theta(\tau_0)$ from another simplified function for the operation frequency defined in (4):

$$f(\tau) = (b \cdot \theta_C / a)^{1/\alpha}. \tag{4}$$

where a, b, α are the same fitting constants as in (3).

The linear relationships of temperature and frequency expressed in (3) and (4) are simplified with fixing one or the other [1]. That is, (3) is derived with a constant frequency f_C and (4) is derived with a constant temperature θ_C. Since temperature keeps changing during run-time, we need a way to estimate it. The estimation is based on the derivative of (3):

$$\Theta'(\tau) = -b \cdot (\Theta(\tau_0) - a \cdot f_C^{\alpha}/b) \cdot e^{-b \cdot \Delta\tau}. \tag{5}$$

where a, b, α are the same as in (3).

III. Temperature-Aware Online Dynamic Multiple Feasible Intervals

In our system, DMFI task sets begin as Earliest Deadline First (EDF) tasks. Feasible intervals of each task are generated later by considering outside factors, current circumstances, and other tasks. For example, a multi-threaded download manager can be required to start a high priority task set immediately. Thus, it needs to adjust feasible intervals of its current, ready tasks besides determining running intervals for this new comer. We previously proposed an efficient feedback-based online scheduling algorithm ODMFI in [26] to address the dynamic features of MFI task sets. It aggressively utilizes slack times by scaling CPU frequencies according to a modeled execution time instead of Worst Case Execution Time (WCET). However, it can also dramatically upscale CPU frequencies when there is a workload burst. For example, when many urgent tasks arrive at the same time, ODMFI increases CPU speed to its highest available operating frequency to meet deadlines. This may result in performance degradation and thus longer execution time and higher timing violations. Although a low CPU frequency indicates low power consumption according to (1) ~ (3), the heat dissipation is more complicated. Studies by Liu et cl. in [13] about the relationship between thermal optimization and energy optimization point out that optimizing energy consumption with DVFS can lead to unnecessary high temperature. Thus, it is critical to set a temperature threshold in our scheduling algorithm.

Besides introducing a temperature threshold, we also need to adjust ODMFI for MPSoC. We choose *multilevel scheduling*, a scheduling framework that has been used in modern Operating Systems (OS) and proposed for multiprocessor and multi-core systems [23, 24, 25]. It enables the task allocation to schedulable components e.g. multi-core processors, which further perform fine-grained scheduling among the composing elements of the components, for example on-chip cores. It meets scheduling requirements of MPSoCs such as aggregate performance, load balancing and resource sharing. ODMFI stated in [26] is for single-core platforms. With the extension of temperature-aware control, it naturally becomes a local scheduler controlling each core. A task allocator is designed to schedule MFI tasks among cores in compliance with requirements of temperature and timing. We allow task migration for both thermal and workload balancing. We describe MT-ODMFI in two parts as follows: task allocator and local scheduler.

A. Task Allocator

We assign MFI tasks to each core based on the following rules: 1) cooler cores have higher priorities to get tasks; 2) cores with lighter workload have higher priorities to get tasks.

Rule 1) requires that we maintain each core's current approximated temperature $\theta_{current}$. As mentioned in the above, we estimate $\Theta(\tau_0)$ from (4) before MT-ODMFI starts, and we initialize each core's $\theta_{current}$ with it. As tasks execute on each core, we need to update their temperatures after each task completes. We modify (5) to approximate temperature change caused by each task run from (6):

$$\Delta\theta_i = -b\cdot(\theta_{current} - a\cdot f^{\alpha}/b)\cdot e^{-b\cdot execution\_time(i)}. \quad (6)$$

where a and b are the same constants as in formula (3), $\theta_{current}$ is each core's temperature before task T_i runs, and execution_time(i) is the actual execution time of T_i.

For rule 2), we express workload on each core in temperature as well. Since each local scheduler, temperature-aware ODMFI (T-ODMFI), can queue more than one task when necessary, we have an expectation about how much each core needs to do based on WCET, and thus we can estimate what temperature each core will be at after they finish their assigned work. Because of the severity of thermal problems mentioned previously, we follow *conservative* rule and thus use WCET in temperature prediction. The expected temperature is expressed as $\theta_{expected}$ and we calculate it from modification of (6) with WCET(j) for the assigned task T_j replacing execution_time(i). Thus, we characterize each core with a real pair ($\theta_{current}$, $\theta_{expected}$).

We also allow tasks to be re-assigned to other cores. It happens when any core reaches the temperature threshold θ^*. Task migration is designed in a simple way: re-submit the un-executed tasks to the task allocator. The task allocator will adjust feasible intervals of these tasks accordingly and get them ready to re-assign.

B. Local Scheduler

We introduce temperature-aware ODMFI (T-ODMFI) as a local scheduler which extends ODMFI with temperature constraints. The extension is straight-forward: temperature threshold θ^* is checked every time after each task finishes. Besides temperature checking, the scheduler aggregates available slacks and proportionally scales down the current

Task Allocator

```
While(task queue is not empty)
{
    Choose the core with smallest tuple (($\theta_{current}$, $\theta_{expected}$) for $T_i$;
    Update $\theta_{expected}$ of the chose core with $\Delta\theta$;
    If($\theta_{expected} \le \theta^*$)
    {
        Assign $T_i$ to the chosen core;
        Release memory related to $T_i$;
    }
    Else goto cooling;
}
cooling: check temperature of each core regularly until one of
them has $\theta_{expected}$ in safe range;
```

T-ODMFI Algorithm

```
While(task queue is not empty)
{
    Locate first un-executed task in queue, $T_i$;
    Search available slack times and decide if $T_i$ or other later
tasks suitable to run;
    Choose scale factor for current CPU frequency;
    Scale CPU frequency;
    Run task $T_i$;
    Update $\theta_{current}$ with $\Delta\theta$;
    Adjust $\theta_{expected}$ with the difference between $\Delta\theta$ calculated
with the actual execution time and the one calculated with WCET;
    Release memory related to $T_i$;
    If($\theta_{current} \ge \theta^*$)
    {
        Set CPU frequency to lowest frequency;
        goto re-assign;
    }
}
re-assign: submit all the un-executed tasks to task allocator;
```

Figure 1. Pseudo Code of MT-ODMFI

operation frequency according to the ratio of the estimated execution time to the sum of the available slacks. Contrast to temperature control, we use the estimated execution time which is calculated from curve fittings of historical data. Thus, the temperature-aware extension does not have any impact on the execution time modeler or the selection of CPU frequency scale factor. The algorithms are outlined in Figure 1.

IV. EXPERIMENT DESIGN ISSUES

Temperature threshold selection is critical for meeting deadlines and performance requirements because the on-chip core is effectively paused by the local scheduler when its $\theta*$ is reached. As mentioned in the above, we follow the conservative rule for thermal related metrics and set $\theta*$ as the following. We first refer to the architecture data sheet. For example, there are two case temperatures T_{case} for PXA255 single-core platform [5]. One is the nominal case temperature from 0°C to 85°C. The other is the extended case temperature from -40°C to 100°C. According to [21], a small difference of operating temperature, i.e. 10 ~ 15°C, can result in a 2X difference in the lifespan of devices [21], so 85°C is chosen. Then we provide a cushion with lowering the temperature by a portion, e.g. 15%, which sets 72°C ≅ (1-15%) · 85°C as system θ^* for PXA255 example.

We also need criteria to evaluate our MT-ODMFI algorithm. One key indicator is the energy efficiency defined in (7) which illustrates how much energy is used as dynamic power, because leakage is expected to account for more than 50% of the overall power consumption for feature sizes below 65nm [20]. Leakage power is quite difficult to measure directly, but we can derive it based on (1) and total energy measured through application life time.

$$P_e = P_{dynamic} / P_{total}. \quad (7)$$

V. Future Works

For experiments of MT-ODMFI, several items are on our agenda for the near future. We plan to investigate the effect of thermal issue in the context of adding new tasks to be scheduled in the system, in particular how the addition of new tasks would affect the temperature change. Besides energy efficiency, we also need to compare non-temperature related aspects including but not limited to: 1) missing deadline ratio; 2) overall energy consumption; and 3) algorithm overhead.

Acknowledgment

This work is sponsored in part by the National Science Foundation under Award No. 0720856.

References

[1] S. Wang and R Bettati, "Delay Analysis in Temp.-Constrained Hard Real-Time Systems with General Task Arrivals," in Proc. IEEE Real-Time System Syposium (RTSS), IEEE Press, Dec. 2006, pp. 323-334.

[2] Y. Xie and W.-L. Hung, "Temperature-Aware Task Allocation and Scheduling for Embedded Multiprocessor Systems-on-Chip Design," Journal of VLSI Signal Processing, 45(3), 2006, pp. 177-189.

[3] Y. Liu, R. P. Dick, L. Shang and H. Yang, "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy," in Proc. IEEE Design, Automation and Test in Europe (DATE), Mar. 2007, pp. 204–209.

[4] T. Chantem, R. P. Dick and X. S. Hu, "Temperature-Aware Scheudling and Assignment for Hard Real-Time Applications on MPSoCs," in Proc. IEEE Design, Automation and Test in Europe (DATE), 2008, pp. 288-293.

[5] Intel PXA255 Processor Data Sheet, www.phytec.com/pdf/datasheets/PXA255_DS.pdf.

[6] Y.-W. Yang and K. S.-M. Li, "Temperature-Aware Dynamic Frequency and Voltage Scaling for Reliability and Yield Enhancement," Asia and South Pacific Design Automation Conference (ASPDAC), 2009, pp. 49-54.

[7] J.-J. Chen, J. Wu and C.-S. Shih, "Approximation algorithms for scheduling real-time jobs with multiple feasible intervals," Journal of Real-Time Systems, pages 155-172, vol. 34, no. 3, Nov. 2006.

[8] C.-S. Shih, J. W.-S. Liu and I. K. Cheong, "Scheduling jobs with multiple feasible intervals," in Proc. IEEE Real-Time Computing and Systems and Applications (RTCSA), 2003.

[9] J. Lin and A. Cheng, "Power-aware scheduling for Multiple Feasible Interval Jobs," in Proc. 15th IEEE-CS International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Beijing, China, Aug. 2009.

[10] J. Hall, J. Lin, and A. Cheng, "Dynamic Multiple Feasible Intervals," in Proc. IEEE-CS Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Stockholm, Sweden, April 13-16, 2010.

[11] Nathan Fisher, Jian-Jia Chen, Shengquan Wang and Lothar Thiele, "Thermal-Aware Global Real-Time Scheduling on Multicore Systems," in Proc. IEEE Real-Time and Embedded Technology and Applications Symposium, San Francisco, CA, April 2009.

[12] S. Borkar, "Design Challenges of Technology Scaling," IEEE Micro, July-August 1999, pp. 23-29.

[13] Y. Liu, H. Yang, R. P. Dick, H. Wang and L. Shang, "Thermal vs Energy Optimization for DVFS-enabled Processors in Embedded Systems," in Proc. IEEE 8th International Symposium on Quality Electronic Design (ISQED), 2007.

[14] G. Paci, P. Marchal, F. Poletti and L. Benini, "Exploring "temperature-aware" design in low-power MPSoCs," in Proc. IEEE Design, Automation and Test in Europe (DATE), 2006, pp. 1-6.

[15] J.-J. Chen, S. Wang and L. Thiele, "Proactive Speed Scheduling for Real-Time Tasks under Thermal Constraints", in Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2009, pp. 141-150.

[16] Y. Xie and W.-L. Hung, "Temperature-Aware Task Allocation and Scheduling for Embedded Multiprocessor System-on-Chip (MPSoC) Design," Journal of VLSI Signal Processing 45, 2006, pp. 177-189.

[17] M. Bao, A. Andrei, P. Eles and Z. Peng, "Temperature-Aware Voltage Selection for Energy Optimization," in Proc. IEEE Design, Automation and Test in Europe (DATE), 2008, pp. 1083-1086.

[18] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir and M. J. Irwin, "Thermal-Aware Task Allocation and Scheduling for Embedded Systems," in Proc. IEEE Design, Automation and Test in Europe (DATE), 2005.

[19] A. K. Coskun, T. S. Rosing, K. A. Whisnant and K. C. Gross, "Temperature-Aware MPSoC Scheduling for Reducing Hot Spots and Gradients," Asia and South Pacific Design Automation Conference (ASPDAC), 2008, pp. 49-54.

[20] M. Santarini, "Thermal integrity: A must for low-power IC digital design," Electronics Design, Strategy, News (EDN), September 15, 2005, pp. 37-42.

[21] R. Viswanath, V. Wakharkar, A. Watwe and V. Lebonheur, "Thermal performance challenges from silicon to systems," Intel Technology Journal, (Q3), 2000.

[22] L. Shang, L-S. Peh, A. Kumar and N. K. Jha, "Temperature-Aware On-Chip Netowrks," IEEE Micro, Vol. 26, Issue 1, 2006, pp. 130-139.

[23] J. Bennett and H. Zhang, "Hierarchical Packet Fair Queuing Algorithms," Proc. ACM SIGCOMM '96, pp. 143-156, Aug. 1996.

[24] P. Goyal, X. Guo, and H. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," Proc. Second Usenix Symp. Operating System Design and Implementation (OSDI '96), pp. 107-122, Oct. 1996.

[25] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini and G. D. Micheli, "Thermal Balancing Policy for Multiprocessor Stream Computing Platforms," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 28, Issue 12, pp. 1870-1882, IEEE Press, 2009.

[26] B. Liu, F. Liu, J. Hall, J. Lin and A. Cheng, "Power-Aware Online Dynamic Scheduling for Multiple Feasible Intervals," Department of Computer Science, University of Houston, Technical Report UH-CS-10-06, 2010.

A Study of Dynamic Navigation with WAVE/DSRC in VANET Environment

Shu-Ping Lu, Kuan-Ming Li and Cheng-Yi Hsieh

Abstract— **This paper presents an integrated architecture for the implementation of dynamic routing/navigation application in VANET environment using WAVE/DSRC. Dynamic navigation re-routes after receiving traffic event from VANET is examined in the test using a WAVE On-board-Unit(OBU) and a WAVE Road-Side-Unit(RSU) as the network node. In this study, the loopback detection method is adopted to detect re-broadcasting packets when a forward request in real-time needed. Finally, tests showed that the real-time traffic messages are delivered efficiently for dynamic navigation application with high mobility situations.**

I. INTRODUCTION

Advances in electronic and computer technologies have accelerated the progresses of Intelligent Transport Systems (ITS) recently. ITS is essentially the merger of developments in computing, information technology and telecommunications coupled to automotive and transportation sector expertise [3], [8], [13]. Vehicles communicate to each other to share some important and emergent information in typical application scenarios. ITS presents cost-effective solutions to a wide range of applications such as electronic toll collection, emergency vehicle notification systems, automatic road enforcement.

In content distribution network(CDN) [1], [2] latency fluctuates in the vehicular environment due to such factors as network instability, weak signal and bandwidth limit. With highly progress of wireless network and broadcasting technologies, however, CDN for vehicle communications has now become realistic than ever before. In the well defined infrastructure, vehicles can receive messages from various information sources. First of all, vehicle can receive data from information center via wireless communication or broadcasting (e.g., 3G, FM, Dedicated Short Range Communications(DSRC) [4], [6] and DVB-T[5]) as shown in Fig. 1. Furthermore car-to-car communication could share information while maintaining its mobility. The wireless network with multi-hop transmission usually has low throughput and high packet drop rate in high speed moving. The advent of Vehicular Ad Hoc Network(VANET) [9], [10] have been envisioned to be useful in road safety and many commercial applications. VANET opens a myriad of possibilities towards sharing and exploiting dynamic information. Traffic conditions can be collected and exchanged on the fly while traveling the roads.

Shu-Ping Lu and Kuan-Ming Li are with Information and Communications Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan. {dolinlu,crush}@itri.org.tw
Cheng-Yi Hsieh is with the Department of Electrical Engineering, ASUSTek Computer INC, Hsinchu, Taiwan. hikaruu@seed.net.tw

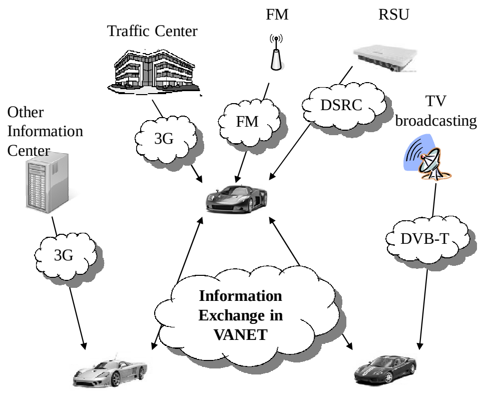


Fig. 1. Overall picture of information services in Vehicle Network.

The way to cope with network dynamics and high mobility efficiently in vehicular network is an important issue. Collectively, as a whole, IEEE 1609.x and IEEE 802.11p are called wireless access in vehicular environments (WAVE) standards because their objective is to facilitate the provision of wireless access in vehicular environments [11]. The real-time road network information may be beneficial for people in moving vehicles to make a better road plan in Dynamic Navigation. However, Yi et al. [12] investigated that the performance of WAVE has not been widely studied. The purpose of this paper is to bring an integrated architecture for the implementation of Dynamic Navigation application in WAVE/DSRC VANET environment. Our results indicate that the real-time traffic messages deliver efficiently for Dynamic Navigation application.

The remainder of this paper is organized as follows: Section II describes system architecture and Dynamic Navigation aspects that are relevant in an environment of WAVE/DSRC VANET wireless transmission. Section III enumerates the throughput analysis of experiments. Finally, Section IV concludes the integrated architecture for the implementation of Dynamic Navigation application in WAVE/DSRC VANET environment.

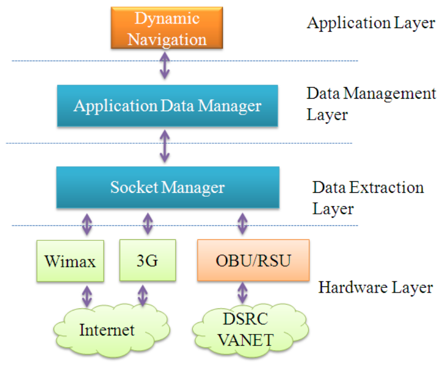


Fig. 2. System Architecture

II. Communication Architecture

In this section, the system architecture and Dynamic Navigation aspects that are relevant in an environment of WAVE/DSRC VANET wireless transmission are described.

A. System Architecture

The aim of our integrated architecture is to evaluate the Dynamic Routing performance over WAVE/DSRC VANET environment. The basic architectural concept of the environment is illustrated in Fig.2.

- *Hardware Layer*: Physical network devices such as 3G, WIMAX, and OBU are defined in Hardware Layer. Vehicles can connect to the Internet through 3G and WIMAX in outer door environment. And vehicles can also establish data communication by inter-vehicle communication by OBU through DSRC protocol.
- *Data Extraction Layer*: Data Extraction Layer extracts service information messages from data received by Hardware Layer. The communication through socket by OBU, 3G, and WIMAX perform socket data extraction in this layer.
- *Data Management Layer*: The data blocks to transferred are stored and extracted from Data Extraction Layer. Application Data Manager broadcasts the data blocks to be transferred via inter-vehicle communication.
- *Application Layer*: Various data applications retrieve data from Application Data Management Layer, such as news, and real-time traffic information.

B. Dynamic Navigation

The components and the data flow of Dynamic Navigation are depicted in Fig. 3. The hybrid model of pre-defined routes is used in Dynamic Navigation to provide the proper path for travelers. For example, the potential traffic jams can be alerted to drivers by vehicular network which providing increased convenience and efficiency. Based on the traffic data, vehicles are also able to calculate the optimal paths. Pre-defined routes select an origin and a destination and

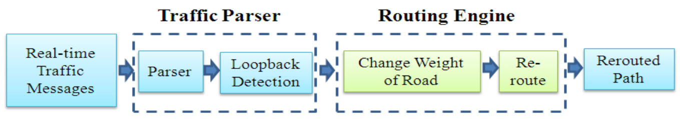


Fig. 3. Data flow of Dynamic Navigation

then calculate the route based on a shortest-path method, such as A* algorithm and Dijkstra's algorithm. Shortest-path based on time uses not only the speed limits of segments, but mainly the dynamic traffic information derived from WAVE/DSRC VANET communication. Furthermore, we develop a loopback detection method to discard the forwarded packet. For example, when Driving car receives the same message again from vehicular environment, the broadcasted message may loop back to the original sender. Once Driving car receives the traffic event from vehicular environment, routing engine changes the weight of accident road section and re-routed. Algorithm 1 shows the establishment of the Traffic Parser with loopback detection.

Algorithm 1: Traffic Parser Procedure

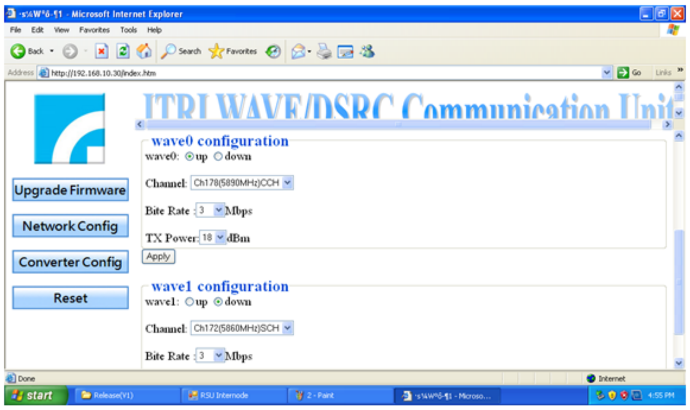
```
input : Traffic message $\{m_1, m_2, ..., m_d\} \in M$, where $m_d$
        with RSU/OBU unique ID $\{w_1, w_2, ..., w_k\} \in W$
output: Packetize message to be forwarded $P$

1  begin
2      $P \longleftarrow \phi$
3      Traffic Buffer $TB \longleftarrow \phi$
4      for $i \leftarrow 1$ to $d$ do
5          $TB \longleftarrow TB \cup m_i$
6          if $w_k$ existed then
7              $TB \longleftarrow \{TB - m_i\}$
8          else
9              $P \longleftarrow TB \bigcup w_k$
10         Broadcast $P$
```

III. Performance Evaluation

A. An implement of DSRC Box

The IEEE has developed a system architecture known as WAVE to provide wireless access in vehicular environments [11]. ITRI WAVE/DSRC Communication Unit(IWCU) can transfer data in WAVE/DSRC VANET environment. ITRI develops ITS applications that utilize WAVE/DSRC for two years. The DSRC Box considers that the next generation of vehicles will be commonly equipped with a wireless communication apparatus, according to IEEE 802.11 standards.

IWCU has two wave ports and Fig. 4 is the setup interface. Each wave port has different RF antenna, channel plan and network converter. We could use web browser to connect



wave0 configuration
wave0: [✓ up] down
Channel: Ch178(5890MHz)CCH
Bite Rate : 3 Mbps
TX Power: 18 dBm
Apply

wave1 configuration
wave1: up [✓ down]
Channel: Ch172(5860MHz)SCH
Bite Rate : 3 Mbps

Fig. 4. ITRI DSRC/WAVE Communication Box Setup

the DSRC box to configure. For channel plan, the setting contains channel, bit rate and TX power. Higher TX power could transmit packet data further but power consumption is also higher. Network converter is the rule how DSRC box converts the Ethernet UDP/IPv4 packet to DSRC network packet. The converter setting contains Provider Service Identifier (PSID), IPv4, UDP Port and Wave Port. One DSRC box which has converter with one PSID number could receive the packet with the PSID number from the other DSRC box. IPv4 is the TCP/IP address of Ethernet packet that requires transform to 802.11p packet. The address also used to transform 802.11p packet to Ethernet packet. UDP port is the port number of TCP/IP when packet is Ethernet. Wave port is the port number when packet is 802.11p.

Because IWCU delivers and broadcasts data in vehicular communication, the IP addresses of all DSRC box are set to a same address as 192.168.10.10. We also set the PSID, UDP port and Wave port to a same number, 123. The application receives all the broadcasting packets and identifies the different service by checking a number in the network message.

B. *Traffic Efficiency*

Vehicular communication is used to create and share the traffic related information. In this paper, traffic efficiency is the criteria to measure Dynamic Navigation performance of the transportation network. The criteria will be used to address the performance of the experimental results later. Table I shows the traffic efficiency of 14 sec response time and 200 meters response distance by the speed of 40km/hr or 60km/hr.

C. *Experimental Results*

The trials [7] were based on an OBU that is only able to provide position information to enable the road segment on which the vehicle is travelling to be identified. The implementation of this paper addressed a Dynamic Navigation application and developed a loopback detection method to discard the re-forwarded packet. After the car received the traffic event in the original routing path, it rerouted to another routing path to avoid road section which had the traffic event. Furthermore, the loopback detection method was adopted to discard the broadcasting packet to be re-forwarded.

TABLE I

TRAFFIC EFFICIENCY OF 14 SEC RESPONSE TIME AND 200 METERS RESPONSE DISTANCE BY THE SPEED OF 40KM/HR OR 60KM/HR.

Speed	40km/hr	60km/hr
Msg Arrive	45s, 526.760419002042m	30s, 552.771699410834m
Route Begin	45s, 526.760419002042m	30s, 552.771699410834m
Route End	43.96s, 516.449716783234m	28.96s, 537.916799130385m

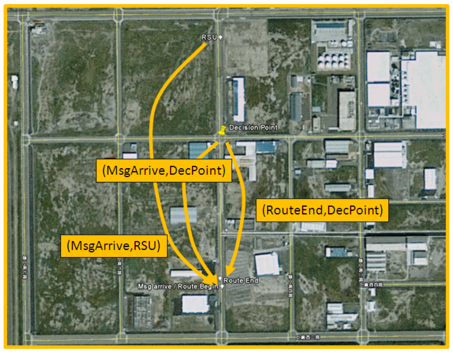


Fig. 5. The real word environment

Fig. 5 shows the real world scenario of the experiment. The top node was the position of Event car with one DSRC Box. The middle node was decision point of Dynamic Navigation. The rest nodes were the positions of message arrival, route end, route begin, and Driving car with one DSRC Box. The routing procedure of this scenario is described in the following and depicted in Fig. 6 and Fig. 7. Note that the bolded line was the routing path. The bolded line was changed while receiving real-time traffic event. In Dynamic Navigation demo, one DSRC Box on Event car beside the road and another one on Driving car were set up. Event car broadcasts the traffic event out. Once Driving car received the traffic event from Event car, routing engine changed the weight of accident road section and re-routed. When Driving car received the same message again, the broadcasted message might loop back to the original sender.

Table II shows the distances between nodes of experimental results. Traffic efficiency of experimental results in response time and response distance are displayed in Table III. Compared with Table I, the results indicate that Dynamic Navigation works well and efficiently in WAVE/DSRC VANET environment when receiving real-time traffic information.

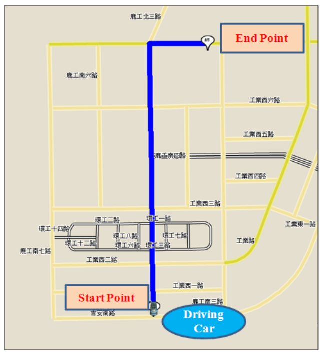


Fig. 6. The original path before rerouting

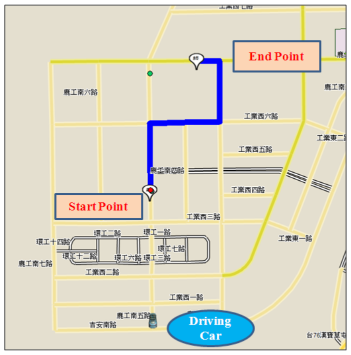


Fig. 7. The re-routed path

IV. Conclusion

In this paper, we proposed an integrated architecture for the implementation of Dynamic Navigation application using WAVE/DSRC in VANET environment. WAVE/DSRC is a low latency and high bandwidth device that is necessary for real-time safety-critical applications, such as collision avoidance or pedestrian safety applications. Dynamic Navigation could re-route after receiving traffic events. The loopback detection method was adopted to eliminate the broadcasting packet to be re-forwarded. The real-time traffic messages delivery for Dynamic Navigation application was examplified.

By the time this work is ongoing, the implemented environment is being used to further investigate various use cases that involves Vehicle-to-Vehicle communication. Future research are planned to assessed a comprehensive VANET applications using this mechanism, and will be conducted in much larger scale.

TABLE II
Distances between nodes

Speed	40km/hr	60km/hr
Routing Time	1.045s	1.029s
(MsgArrive,RSU)	907.36394326477m	948.89419810080m
(RouteBegin,RSU)	907.36394326477m	948.89419810080m
(MsgArrive, DecPoint)	526.76041900204m	552.77169941083m
(RouteBegin, DecPoint)	526.76041900204m	552.77169941083m
(RouteEnd, DecPoint)	516.44971678323m	537.91679913038m

TABLE III
Traffic Efficiency of Experimental Results in Response Time and Response Distance

Speed	40km/hr	60km/hr
Msg Arrive	24.0680483333333s, 120.393166666667m	24.0676733333333s, 120.393168333333m
Route Begin	24.0680483333333s, 120.393166666667m	24.0676733333333s, 120.393168333333m
Route End	24.0681405008717s, 120.393115839189m	24.0679463546302s, 120.393089956913m
Decision Point	24.0728038333332s, 120.393037166892m	24.0728038333332s, 120.393037166892m

References

[1] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[2] M. Bateni and M. Hajiaghayi. Assignment problem in content distribution networks: unsplittable hard-capacitated facility location. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–814, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[3] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. Ibm infosphere streams for scalable, real-time, intelligent transportation services. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pages 1093–1104, New York, NY, USA, 2010. ACM.

[4] J.-M. Bohli, A. Hessler, O. Ugus, and D. Westhoff. A secure and resilient wsn roadside architecture for intelligent transport systems. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 161–171, New York, NY, USA, 2008. ACM.

[5] P. Bures. The architecture of traffic and travel information system based on protocol tpeg. In *EATIS '09: Proceedings of the 2009 Euro American Conference on Telematics and Information Systems*, pages 1–8, New York, NY, USA, 2009. ACM.

[6] J. He, H.-H. Chen, T. M. Chen, and W. Cheng. Adaptive congestion control for dsrc vehicle networks. *Comm. Letters.*, 14(2):127–129, 2010.

[7] M. Kitchen and S. Hoepfel. *Traffic Choices Study V Puget Sound Trial*. ITS (UK) Road User Charging Interest Group, November 2005.

[8] S. Kurihara, H. Tamaki, M. Numao, J. Yano, K. Kagawa, and T. Morita. Traffic congestion forecasting based on pheromone communication model for intelligent transport systems. In *CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, pages 2879–2884, Piscataway, NJ, USA, 2009. IEEE Press.

[9] J. Lee, G.-L. Park, S.-W. Kim, H.-J. Kim, and S. Y. Shin. A hybrid prefetch policy for the retrieval of link-associated information on vehicular networks. In *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 189–193, New York, NY, USA, 2010. ACM.

[10] O. K. Tonguz and M. Boban. Multiplayer games over vehicular ad hoc networks: A new application. *Ad Hoc Netw.*, 8(5):531–543, 2010.

[11] R. A. Uzcátegui and G. Acosta-Marum. Wave: a tutorial. *Comm. Mag.*, 47(5):126–133, 2009.

[12] B. K. Yi Wang, Akram Ahmed and K. Psounis. Ieee 802.11p performance evaluation and protocol enhancement. *Proceedings of the 2008 IEEE International Conference on Vehicular Electronics and Safety*, pages 317–322, 2008.

[13] T. Zelinka and M. Svitek. Identification of communication solution designated for transport telematic applications. *WTOC*, 7(2):114–122, 2008.

Development of Mesovirtualization for the ARM Architecture

Akihiro Suzuki Shuichi Oikawa
Department of Computer Science
University of Tsukuba
Tsukuba, Japan

***Abstract*—The performance gain of embedded processors is significant in theses days. As their performance is being increased, it is very likely that embedded systems will use virtualization technologies. This paper describes the development of SIVARM, a virtual machine monitor (VMM) for the ARM architecture, which is the most popular processor architecture among embedded systems. An obstacle to introduce virtualization to embedded systems is high development costs of VMMs. In order to overcome the obstacle, we applied mesovirtualization to its development, so that we could simplify the development effort. After less than a year of our effort to develop SIVARM, we could successfully execute Linux on SIVARM. We can now perform some experiments on an emulated environment using QEMU, which defines the Integrator/CP board with the ARM926EJ-S processor. We are currently working on porting SIVARM to the ARM1136JF-S (Freescale i.MX31) based system.**

***Keywords*-Virtual machine monitors; embedded systems; ARM architecture**

I. Introduction

The performance gain of embedded processors is significant in theses days. The clock frequency of some of such processors reaches 1 GHz. Also, the trend of CMP (Chip Multi-Processing) or multi-core architectures diffuses silently, and processor manufacturers have begun shipping of such products. With the high performance of current embedded processors, which is comparable to the PCs of a few years ago, it is very likely that embedded systems will use virtualization technologies. Virtualization enables more dependability and higher security [3], [5], and those features are beneficial to embedded systems, too.

An obstacle to introduce virtualization to embedded systems is high development costs of virtual machine monitors (VMMs) [6]. VMMs are the software layer that runs on bare machines and creates virtual machines, on which operating systems (OSes) can run. Most VMMs have been developed targeting servers, and many of them are for the IA-32 architecture because of its popularity. There are few attempts to port full fledged VMMs to embedded processors, and XenARM [7] is such an example. XenARM is a port of the Xen hypervisor [1] to the ARM architecture. Its development is, however, very slow. After more than 3 years of its development, it is not still usable enough even for experimental uses.

This paper describes our effort to develop a VMM, called SIVARM, on the ARM architecture. We apply mesovirtualization to its development. Mesovirtualization is a lightweight virtualization technique, which we developed for the IA-32 architecture [8]. Mesovirtualization can make VMMs smaller and require only a few modifications for the guest operating system (OS) source code; thus, we can keep the development cost of our VMM as low as possible. After less than a year of our effort to develop SIVARM, we could successfully execute Linux on SIVARM. We can now perform some experiments on an emulated environment using QEMU[2], which defines the Integrator/CP board with the ARM926EJ-S processor. We are currently working on porting SIVARM to the ARM1136JF-S (Freescale i.MX31) based system.

II. Mesovirtualization

Mesovirtualization is a lightweight virtualization technique to construct a VMM. Mesovirtualization opts to modify a few parts of the guest OS source code in order to enable lightweight configuration of a VMM, but the cost of modifications can be kept as low as possible to make the modifications easily manageable. Therefore, it does not require the complicated work typically needed for full virtualization and paravirtualization. We must configure huge VMMs for full virtualization and modify a huge amount of the guest OS source code for paravirtualization. Mesovirtualization does not require such complicated work, yet it can provide guest OSes with sufficient virtualization environments, in which guest OSes can manage their environments, use processors, memory and devices as if they run on a physical machine.

Mesovirtualization is based on the principle of minimalism. We do not need to virtualize the entire of the host machine to provide identical environments to guest OSes as full virtualization. We do not need to modify many parts of the guest OS source code to trap into a VMM and to handle it in the VMM as paravirtualization. Mesovirtualization is a technique which supports guest OSes just enough to run it on a VMM. For some parts of the host machine that are considered safe to be dedicated or shared, a VMM does not virtualize these parts and allows guest OSes touch them directly. This rationale keeps a VMM as simple as possible. It is beneficial to the development of the both VMM and guest OSes in a sense that the code size and memory footprint of the VMM becomes small and also that the costs of virtualization can be kept cheap. Therefore, it works better on embedded and ubiquitous systems, of which computing resources are not as rich as desktop and server systems.

III. ARM Architecture

This section describes the features of the ARM architecture that are different from the IA-32 architecture and that affect the design and implementation of SIVARM.

The processor mode of the ARM architecture is composed of the privileged mode and the non-privileged mode (USR), and there are 6 different types of the privileged mode, FIQ, IRQ, SVC, ABT, UND and SYS, corresponding to different reasons to trap into the the privileged mode.

Among those types of the privileged mode, 5 of them except for SYS are for handling exceptions and interrupts. In order to avoid saving registers and to enable quick handling of exceptions and interrupts, the ARM architecture employs banked registers. FIQ defines 5 general purpose registers, a stack pointer register (SP), a link register (LR), and a saved program status register (SPSR) in its banked registers. The other 4 types defines only SP, LR, and SPSR in their banked registers. When an exception or interrupt occurs, the corresponding banked registers are automatically used.

Table I shows the reasons to trap into the the privileged mode and the types of the privileged mode used to handle traps. The ARM architecture has 7 exceptions. When an exception occurs, the processor mode changes to the corresponding exception type, the CPSR is saved in the SPSR of its banked registers, and the execution starts from the fixed address according to the caused exception type.

The ARM architecture employs a two-stage access control using a domain and an access permission (AP) bit. The domain can divide memory space into several parts. When a virtual memory space is shared, it can isolate a part of memory space from the other. The AP bit controls access to a specific virtual memory page frame based on the current processor mode. Both the domain ID and the AP bit are included in page table entries.

Sensitive instructions are the instructions that affect or depend on the processor state, and a VMM needs to handle them appropriately when a guest OS kernel executes them. The ARM architecture includes several sensitive instructions. Most of them are privileged instructions, of which execution in the non-privileged mode causes an exception. There are, however, a few sensitive instructions that can be executed in the non-privileged mode. The access instructions, MRS and MSR, to the current program status register (CPSR) are such examples.

IV. Design and Implementation

This section describes the design and implementation of SIVARM and the modification needed to the Linux kernel, which we use as a guest OS to run on SIVARM.

A. Virtual Processor Mode

Since the ARM architecture provides only the two execution modes, the privileged mode and the non-privileged mode, there is no choice other than executing a VMM in the privileged mode. In order to protect a VMM from a guest OS, both the kernel and its user processes of the guest OS need to run in the non-privileged mode. This way, however, introduces a problem. Since the Linux kernel expects itself to run in the privileged mode, it has to be able to distinguish in which of the 6 types of the privileged mode it is currently running.

We solve the problem by introducing the virtual processor mode. SIVARM creates an illusion that there are virtually the 7 types of the privileged mode and the non-privileged mode. We execute kernel mode of Linux in virtual privileged mode and user mode of Linux in virtual non-privileged mode.

B. Virtual Banked Registers

The virtual processor modes must have the virtual banked registers as the real processor modes. But, the number of registers is limited in non-privileged mode; thus, extra registers do not exist.

We use the memory space of SIVARM as the space of the virtual banked registers. The virtual banked registers prepare the registers that are used in virtual USR and virtual SYS. In this study, we call it the virtual registers. And, when the guest OS executes the instruction of changing the processor mode, we store the current registers to the virtual registers and copy the virtual banked registers to the current registers. These can use the current registers as the banked registers.

Moreover, the CPSR before the exception is stored to the SPSR of the mode of the caused exception automatically in the real processor modes. So, in the virtual processor modes too, the CPSR before the exception is stored to the virtual SPSR of the virtual mode of the caused exception by the hand.

C. Virtual Protection Level by Domain

We must allow the virtual privileged mode to access all memory space in the guest OS. And, we must not allow the virtual non-privileged mode to access the kernel memory space in the guest OS. These are the same protection of the real processor modes. In addition, SIVARM is executed in privileged mode of the real processor mode. So, we must maintain the state that the VMM is protected from accessing by the guest OS.

Therefore, in this study, we construct a virtual protection level by using the domain. We use the domain as shown in Table II. D0, D1 and D15 show respectively the domain where the kernel process belongs, the user process belongs and SIVARM belongs. It is necessary that the AP bit of the PTE that the VMM belongs is set to 0b01 beforehand, and not to access the VMM from non-privileged mode. Moreover, D15 that the VMM belongs is always set to the

Table I
THE EXCEPTION TYPE, THE PROCESSOR MODE AND THE HIGH VECTOR ADDRESS

Exception Type	Type of Privileged Mode	Vector Address
Reset	SVC	0xFFFF0000
Undefined instructions	UND	0xFFFF0004
Software interrupt (SWI)	SVC	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	ABT	0xFFFF000C
Data Abort (data access memory abort)	ABT	0xFFFF0010
IRQ (interrupt)	IRQ	0xFFFF0018
FIQ (fast interrupt)	FIQ	0xFFFF001C

Table II
THE ACCESS CONTROL FOR VIRTUAL PROCESSOR MODE USING DOMAIN.

	D15	D1	D0
VMM (D15)	Client	Manager	Manager
User (D1)	Client	Client	No access
Kernel (D0)	Client	Manager	Manager

client, and the access control always uses the AP bit. As a result, the VMM is protected from the guest OS that is executed in non-privileged mode.

As mentioned above, the virtual protection level is constructed by using the domain in non-privileged mode, and Linux can be executed as the guest OS.

D. Exception Handling

Exceptions need to be handled by Linux while they are first delivered to SIVARM since the ARM architecture defines the fixed start addresses of exception vectors. SIVARM stores appropriate branch instructions at the exception vectors during the initialization, so that exceptions invoke SIVARM's exception handlers. SIVARM's exception handlers then calls Linux's exception handlers. Since Linux assumes exception handlers run at the corresponding processor modes, he virtual banked registers described in Section IV-B are used.

Depending up the source of exceptions, SIVARM determines if exceptions need to be delivered to Linux or not. If the source of exceptions are Linux's user processes, they are delivered to Linux. If the source of exceptions are the Linux kernel, SIVARM examines the exceptions in order to determine if SIVARM needs to handle them. If they are caused by the execution of sensitive instructions in the Linux kernel or device accesses, SIVARM handles them. Otherwise, they are delivered to Linux.

E. Emulation of Non-Privileged Sensitive Instructions

Both the MRS and the MSR instructions are non-privileged instructions; thus, their execution in user mode does not cause an exception. They are, however, sensitive instructions, which SIVARM needs to detect their execution. If they execution cannot be detected, SIVARM cannot execute the guest OS correctly.

In this study, we rewrite the MRS and the MSR instructions in the code of Linux to an adequate privileged instruction off-line. The instruction obviously causes an exception, so SIVARM detects the execution of the non-sensitive instruction using the substitutional privileged instruction. In this study, we call the instruction causing an exception on purpose the representative privileged instruction. We use following privileged instruction as the representative privileged instruction. It's operand isn't used in Linux.

```
mrc p15, 7, rX, c7, c7, 7
```

It is necessary for the VMM to associate the representative privileged instruction with the non-privileged sensitive instruciton.

The representative privileged instruction allows SIVARM to emulate the MRS and the MSR instructions in the guest OS. The exception caused by the representative privileged instruction is the undefined instructions exception. The way of the emulation is the same until the PC moves to the undefined exception handler. The undefined instructions exception handler in the VMM describes the handling of the representative privileged instruction individually in advance.

The representative privileged instructions enable the correct emulation of the non-privileged sensitive instruction.

V. EXPERIMENT AND EVALUATION

This section describes the experiment and evaluation done on SIVARM constructed based on the design and implementation policy described in section IV.

A. Experiment Environment

The experimental environment of this system is as follows. We developed and evaluated SIVARM on an emulated environment using QEMU.

- OS : Linux 2.6.25.4
- Board : Integrator/CP
- CPU : ARM926EJ-S (ARMv5TEJ)
- Memory : 128MB

B. Evaluation of the Size of Implementation

When SIVARM was constructed based on the design and implementation policy described in Section IV, it became 2662 lines in C and 792 lines in assembly language including inline assembler. The representative privileged instructions

Table III
EXECUTION RESULT OF EACH BENCHMARK

Benchmark	NativeLinux (usec)	SIVARM (usec)	Speed ratio
fork+exit	3,255.97	86,488.48	26.56
fork+exec	14,917.65	274,880.32	18.43
pipe	210.02	1,001.09	4.77
syscall	0.87	7.21	8.29

described in Section IV-E are 44 lines (the number of the MRS instruction is 20 lines and the MSR instruction is 24 lines) in total. So, the change to the Linux kernel is very little. In addition, we adopt that the way of booting SIVARM by adding the VMM to the zImage in this study. So, we add 70 lines to the decompression handling of the zImage.

C. Evaluation of the Performance

We executed a benchmark program on Linux introducing SIVARM and native Linux. It was done with the execution time of each instruction have been made constant by using *icount* option of QEMU.

We show the result of comparing the execution speeds in Table III. The speed ratio shows the execution speed of SIVARM when the execution speed of Native Linux is adjusted to one.

According to Table III, an intense performance decrease occurs in SIVARM. Therefore, it is not possible to use the VMM by the scene where real time is demanded at present. We will aim at the performance gain of the VMM in the future. Especially, the speed ratio of the benchmark that creates a new process by using fork like fork+exit and fork+exec is higher than the other one. It seems that this is because a lot of privileged instructions are executed for PTE when PT is updated when the process is created and the number of emulation executed in the VMM increased.

VI. RELATED WORK

Xen[1] is a popular VMM that runs on the IA-32 architecture. It was also ported to the ARM architecture, and is called XenARM [7]. XenARM employs paravirtualization, so that a significant amount of modifications are necessary to its guest OS kernel; thus, Its development is, however, very slow. After more than 3 years of its development, it is not still usable enough even for experimental uses. On the other hand, SIVARM employs mesovirtualization, which requires only a small amount of changes to its guest OS kernel. After less than a year of our effort to develop SIVARM, we could successfully execute Linux on SIVARM.

The ARM architecture now provides a hardware assisted virtualization feature, called TrustZone which is available only on few high end processors, such as ARM1176JZF-S. By taking advantage of TrustZone, virtualization should become easier as it happened when the Intel VT was introduced to the IA-32 architecture. The ARM architecture is, however, used in a variety of SoC solutions; thus, such a hardware feature may not be always available. Therefore, a lightweight VMM like SIVARM is useful for those solutions.

VII. SUMMARY

This paper described the development of SIVARM, a VMM for the ARM architecture. We applied mesovirtualization to its development, so that we could simplify the development effort. After less than a year of our effort to develop SIVARM, we could successfully execute Linux on SIVARM. We can now perform some experiments on an emulated environment using QEMU. We are currently working on porting SIVARM to the ARM1136JF-S (Freescale i.MX31) based system. In addition, we are trying to improve the performance of a guest OS on SIVARM.

REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield: Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, pp. 164-177 (2003).

[2] Febrice Bellard: QEMU, a Fast and Portable Dynamic Translator, USENIX 2005, pp.41-46 (2005).

[3] T. Bressoud and F. Schneider: Hypervisor-Based Fault Tolerance. *ACM Transactions on Computer Systems*, Vol.14, No.1, pp.80-107 (1996).

[4] M. Cereia, I. C. Bertolotti: Virtual Machines for Distributed Real-Time Systems, Computer Standars & Interfaces, pp.30-39 (January 2009).

[5] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh: Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 193-2006 (2003).

[6] R. P. Goldberg: Survey of Virtual Machine Research, IEEE Computer, Vol.7, No.6, pp.34-45 (1974).

[7] Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, Chul-Ryun Kim: Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones, CCNC 2008, pp.257-261 (2008).

[8] M. Ito, S. Oikawa: Mesovirtualization: Lightweight Virtualization Technique for Embedded Systems. In *Proceedings of the 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems*, pp. 496-505 (2007).

Handling Lock-Holder Preemption in Real-Time Virtualization Layer for Multicore Processors

Hitoshi Mitake, Yuki Kinebuchi, Alexandre Courbot, Tatsuo Nakajima
Department of Computer Science
Waseda University
{*mitake, yukikine, alex, tatsuo*}*@dcl.info.waseda.ac.jp*

***Abstract*—Porting ordinary operating systems to a virtual machine monitor produces a semantic gap because assumptions which guest OSes rely on cannot be preserved. On multicore environments, this gap can cause fatal performance degradations. Lock Holder Preemption (LHP) is a well known example of the sources of perfromance degradation. It occurs when a thread of guest OS holding a mutex based on busy wait is preempted by virtual machine monitor.**

Some previous research developed techniques to avoid this phenomenon, but none of them cares about real-time responsiveness of guest OSes. In this paper, we introduce a plan for developing a new method to avoid LHP. The difference between previous methods and our new one is awareness of real-time responsiveness of guest OSes. So this method can be applied to virtualization technology designed for real-time embedded systems.

I. Introduction

As predicted by Moore's law, today's computer systems have become significantly powerful especially in number of cores. The increase of processor cores turned virtualization technology into a real world method. In server side computing, virtualization already became a de-facto standard method, mainly for system integration. In this field, virtualization significantly reduces cost of engineering, management, and hardware resources.

Increasing the number of processor cores is becoming a common trend for embedded systems too. Rich devices like smart phones have become functional along with hardware enhancements. However developing dedicated software for each rich device introduces a siginificant engineering cost.

The increasing cost of developing and managing software is a problem that not only embedded systems but also servers have to address. Of course, both sides have their own specific problems. In this work, we will concentrate on the specific problems of real-time responsiveness for virtualized embedded systems.

Guaranteeing real-time responsiveness is one of the representative problems of embedded systems. In addition, recent embedded models feature new and rich interfaces like touch panels. There is therefore a difficult convergence point to be reached in today's embedded systems: featuring both common applications with rich interfaces and real-time responsiveness on the same device.

This goal is very hard to achieve using existing OSes because almost every operating system is one of these two types:

General Purpose OS (GPOS) which provides fair scheduling, high throughput, and highly abstracted interfaces for ease of programming and isolation but no real-time guarantee.

Real Time OS (RTOS) featuring real-time scheduling, low latency, but low abstractions in order to preserve low overhead.

An example of GPOS, Linux, most used OS especially in server side computing, has lots of existing software resource in form of programs, libraries and kernel modules: device drivers, file systems, network protocol stacks, natural language processors, multimedia players and so on. But, as a Unix-like operating system, Linux is not suitable for running real-time processes: for instance, a non real-time process can block the interrupts globally and thus prevent execution of real-time processes within a non-predictable timespan.

On the other hand, a traditional RTOS is suitable for running real-time processes, but misses the huge software library that OSes like Linux have.

For these reasons, developing modern embedded devices with both rich interfaces and guaranteed real-time responsiveness based on single GPOS or RTOS is a difficult task. But running the two types of OSes on the same device is a good technique to combine the best of both worlds.

RTLinux [3], Adeos [2] and our ongoing project SPUMONE [1] all propose a solution to combine RTOS and GPOS. However all of them are sharing the same problem according to the semantic gap between real and virtual multicore processor. OSes assume they are controlling hardware resources directly. When porting such OSes to a virtualization layer, a semantic gap appears. In this case, by combining a RTOS and a GPOS, the GPOS loses complete control of the interrupt controller. Therefore there is a possibility for the RTOS to preempt the GPOS even when it is executing a critical section. This preemption can cause critical performance degradation of GPOS, because if other threads of GPOS running on a different CPU try to acquire a lock that is already acquired by the preempted thread, they spin in vain until the RTOS releases CPU.

Preempting the virtual CPU which is executing a thread holding a lock is a common problem of virtualization technology and is referred to as Lock Holder Preemption [4]. Previous research [4], [5] proposed some solutions to this problem, but as we will describe in this paper, these techniques cannot be applied to real-time systems.

In this work, we first demonstrate the problem and try to establish a method also applicable to real-time systems.

II. Brief Description of SPUMONE

The method we are trying to establish is for a virtual machine monitor called SPUMONE [1]. This is a para-virtualization technology that works as a thin abstraction layer between hardware and OSes. Virtualization technology categorized into para-virtualization requires modification of guest OSes, but overhead of the virtualization layer is relatively low.

In SPUMONE, each OS runs on a vCPU provided and scheduled by SPUMONE. The vCPU scheduler is activated by interrupt and through the sleep instruction of guest OSes. When the guest OSes issue a sleep instruction, SPUMONE selects the next runnable vCPU assigned to its physical CPU (pCPU). If there is no runnable vCPU, a special vCPU representing an idle state is executed just like the idle task of traditional OSes. And when interrupt is raised, SPUMONE delivers it to the vCPU which the interrupt number is assigned to.

SPUMONE leverages the interrupt priority mechanism of hardware. Thanks to this mechanism, it is possible to mask interrupts partially. The interrupt controller judges which interrups should be masked based on their priority. For example, if we assume that there are two devices with different priorities, one being the timer device which rises an interrupt periodically and the other being the ethernet controller which rises an interrupt when it receives a frame, if a high priority is assigned to the timer device and a low one to the ethernet controller, the interrupt service routine of the OS processing the interrupt of the timer device can mask the interrupt of the ethernet device.

This feature is key to implementing SPUMONE. In the example described above, if the RTOS is assigned the timer device and GPOS the ethernet device, SPUMONE delivers timer interrupts to the vCPU of the RTOS, triggering its execution. If the ethernet controller receives a frame and tries to raise an interrupt during the execution of the RTOS, the pCPU will not be interrupted by the ethernet controller because the priority of the timer device is higher than the one of the ethernet controller.

This interrupt priority mechanism is a common feature of CPU targeting embedded systems. Our test environment, SH-4A, provides it, as well as ARM-based processors.

As the description above implies, SPUMONE requires modifying the source code of guest OSes. The amount of modification is however very small: only the entry of interrupt service routine and functions for issuing sleep instruction.

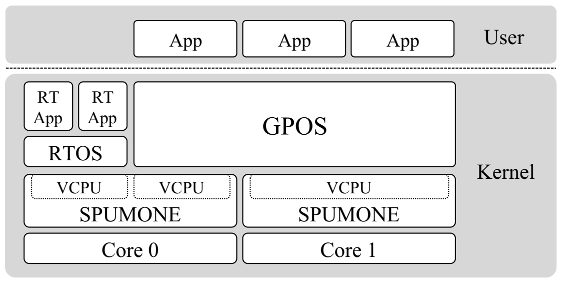


Figure 1. Structure of SPUMONE on multicore processor

There are several similar work focusing on combining a RTOS and a GPOS together. RTLinux [3] treats Linux as a idle task of the real-time OS. Linux is therefore allowed to run when there is no real-time tasks to run. Adeos [2], targeting the x86 architecture, enables running multi OSes on one system without modification of guest OSes. It only requires inserting kernel module, but highly depends on the unused privilege levels of the x86 architecture.

III. Problem Statement: Multicore Specific Performance Problem

SPUMONE is a successful technology on single core environment. But on multicore systems, depicted in Fig.1, a critical performance degradation of the GPOS is observed.

Fig.2 is the result of running the backbench benchmark. The leftmost bar, labeled 4 cores, is the score of Linux running on 4 dedicated physical cores. The rightmost bar, labeled 3 cores, is the score of Linux running on 3 dedicated physical cores. The bars in the middle, labeled with percentages, are the scores of Linux running on top of 4 physical core while sharing one core with the RTOS. The percentage describes the CPU time consumed by the RTOS.

As this graph describes, when the CPU time consumption of the RTOS is larger than 70%, the scores of hackbench is lower than when Linux is running on 3 cores only.

This performance degradation occurs because of the Lock Holder Preemption (LHP) [4] phenomenon. In general, LHP is caused by preemption of the vCPU executing a thread holding a busy-wait mutex (e.g. spinlock).

LHP causes a significant waste of time because when other vCPUs try to acquire the lock acquired by preempted vCPU, the vCPU holding the lock cannot continue to execute the critical section protected by the lock because it is preempted. Therefore, other vCPUs trying to acquire the lock will spin in vain until the blocked vCPU is scheduled again and finally releases the lock, provoking the observed performance degradation.

There is also another problem related to LHP. OSes running on a multicore processor use Inter Core Interrupt (ICI) for some types of synchronization. For example, ICI is

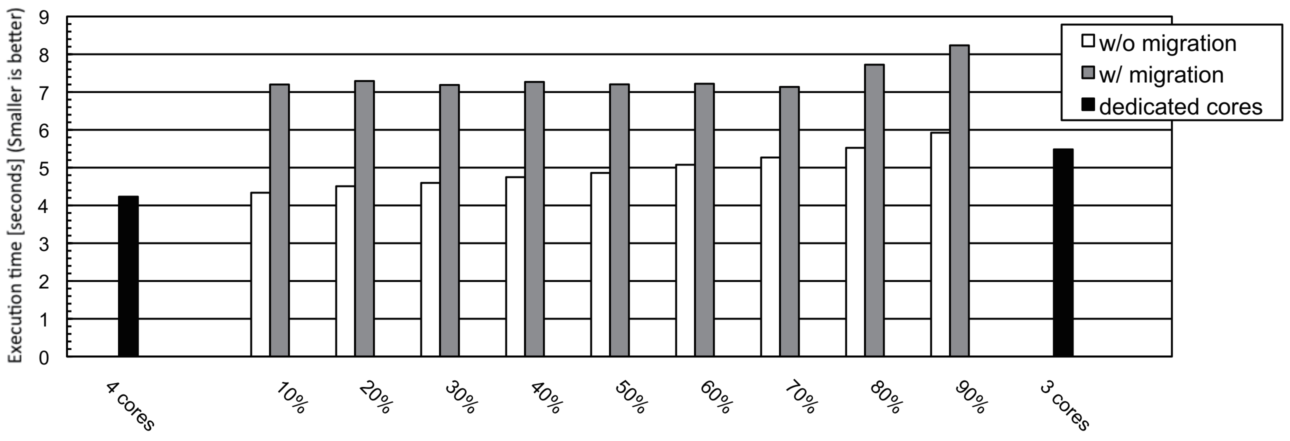


Figure 2. Result of hackbench on multicore SPUMONE

used for TLB shootdown. In a virtualized environment, this can cause significant performance degradation or deadlock if the destination vCPU is preempted.

IV. Related Work for Solving LHP

The traditional way to solve the LHP problem is coscheduling [6]. Coscheduling was originally designed for multi process programs frequently interacting with each other via IPC. A scheduler implementing this method tries to execute processes communicating with each other in same time slice, so that occurrence of blocking by IPC and overhead of context switch is reduced. Today, coscheduling is used for scheduling virtual machines to avoid LHP. For example, VMWare [7] employs a customized version of this method. However, coscheduling still wastes a non-negligible amount of CPU resource, so some projects explored other ways to solve this problem.

Uhlig et al. proposed a method to avoid LHP in both para-virtualization and full virtualization in [4]. The method for para-virtualization is indicating the holding of a lock by a guest OSes with flags of the VMM. If the flags indicate the thread of a guest OS is holding a lock, the VMM never preempts the guest OS but sets another flag indicating delayed preemption. When the thread of the guest OS releases the lock, it checks the delayed preemption flag, which, if turned on, yields the CPU to another VM. This method is efficient for virtualization environments containing GPOSes only, but cannot be applied to RTOSes because it causes delay in interrupt delivery.

Wells et al. proposed another method to avoid LHP using extended hardware in [5]. The extended hardware component, called *Spin Detection Buffer* (SDB), features eight content-addressable memory entries that can hold unique stores and loads instruction. During a given period it records stores and loads. If the entries are not full, SDB indicates that a thread of a guest OS is holding a lock.

All these work introduces an extra delay to the real-time responsiveness of guest OSes, so these techniques can not be applied in a satisfactory manner to embedded systems.

V. Discussion

A. Goal of This Work

The aimed goal of this work is realizing all the conditions below:

Guaranteeing real-time responsiveness of the RTOS,
Preserving the high throughput of the GPOS,
Maximizing CPU resource utilization.

It is not self-evidence whether realizing these three goals at once is possible or not.

For example, when SPUMONE allocates 1 pCPU for the RTOS and 3 pCPU for the GPOS statically, realizing real-time responsiveness and good GPOS throughput without degradation is possible. But this way sacrifices the exploitation of the CPU resource. Because the GPOS can only use 3 pCPU even if the RTOS is sleeping and doing nothing. For exploiting the CPU resource efficiently, effective sharing of a single pCPU with several vCPUs must be realized.

The essential part of this problem is the possibility to schedule an *overcommited* virtual machine. When a virtual machine monitor executes N vCPUs on M pCPUs and $N > M$, the state of the virtual machine monitor is called *overcommitted*.

B. Former Approach

We are trying to solve this scheduling problem using the vCPU migration feature of SPUMONE. SPUMONE can migrate a vCPU from pCPU$_N$ to pCPU$_M$ ($N \neq M$) dynamically. Using dynamic vCPU migration, it is possible to migrate a vCPU of the GPOS from the pCPU executing the vCPU of the RTOS. So the problem is resumed to scheduling overcommited vCPUs of GPOS.

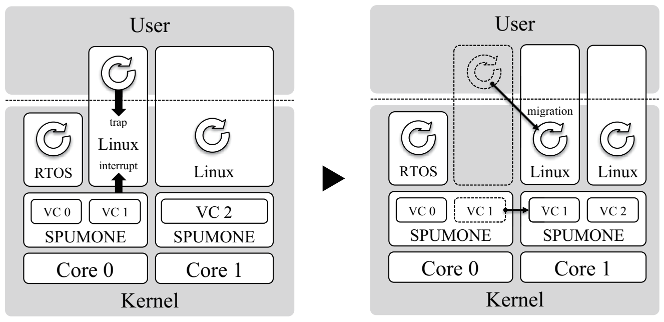


Figure 3. Migration triggered by issuing system call or interrupt

The important difference between this work and related work addressing the LHP is the type of overcommitment. The typical overcommitted state is that the sum of vCPUs number belonging to *several* virtual machines is larger than those of pCPUs. But in our situation, the sum of vCPUs number belonging to a *single* virtual machine is larger than pCPUs.

Current multicore SPUMONE migrates the vCPU of the GPOS running on the pCPU shared with the RTOS when the thread running on the vCPU of GPOS issues a system call or when an interrupt is raised to the vCPU of GPOS in order to avoid LHP as shown in Fig.3. If no thread executing kernel code runs on the shared pCPU, there is no possibility of LHP because the lock with busy wait is virtually only acquired in kernel space.

This approach has the drawback of being too pessimistic. Even if the vCPU of the GPOS does not execute a critical section, its virtual machine will be overcommitted.

C. Directions for New Method

Since the former approach is too pessimistic, we are trying to optimize it in order to reduce frequency of migration.

When LHP occurs, the thread trying to acquire the lock which is acquired by the thread running on preempted vCPU vainly consumes CPU time as lock mechanisms using busy wait, like the spinlock, only do simple iteration when acquiring. So we want to modify the guest GPOS in order to count how long it is waiting for the lock.

If we can define the appropriate threshold of spinning count, guest OSes can recognize occurrence of LHP when the spin count becomes larger than the threshold. This is the appropriate timing to migrate preempted vCPU. The GPOS should request to migrate the vCPU which is currently preempted by the vCPU of RTOS to another pCPU to SPUMONE.

The next problem is scheduling of overcommitted vCPUs running the GPOS only. Because the vCPUs are overcommitted, there is at least one vCPU not executed at a given time. If other vCPU tries to acquire the lock hold by the preempted vCPU, LHP can occur again even on a single OS. But LHP in single OS can also be solved using the scheme described above. The factor that may cause serious problem in this situation is TLB shootdown. If the processors are not always available assumption of ordinary OSes can be violated. In this situation performance can be degraded or deadlock can occur. TLB shootdown employs ICI to notify another CPU to invalidate or update TLB entries. When OSes are executed in a virtualized environment, the vCPU expected to receive the TLB shootdown might not be running. In such case, the vCPU issuing the TLB shootdown cannot obtain any response till preempted vCPUs are resumed and reply. This situation can cause performance degradation. Or if the virtual machine employes fixed priority scheduling for vCPU scheduler, deadlock might occur. We therefore have to modify ICI usage of the GPOS to adopt overcommitted environment.

The last problem is the cost of returning a migrated vCPU to its original pCPU. The policy for migrating the vCPU of the GPOS to its original pCPU right after the vCPU of RTOS sleeps may not work well when a large quantity of interrupts for the RTOS are raised in a short period because of the cost of vCPUs migration. This timing should be decided based on heuristic patterns of the RTOS workload. Whether a dynamic or static policy should be employed is an important design problem. If we employ a dynamic policy, its design becomes another discussion point.

If all of these problems are solved, we should be able to establish a method to handle overcommited virtualization environment while guaranteeing real-time responsiveness.

REFERENCES

[1] Kinebuchi, Y. and Morita, T. and Makijima, K. and Sugaya, M. and Nakajima, T. Constructing a Multi-OS Platform with Minimal Engineering Cost. In *Analysis, Architectures and Modelling of Embedded Systems, 2009*

[2] K. Yaghmour. Adaptive domain environment for operating systems. Opersys inc, 2001.

[3] Victor Yodaiken. The RTLinux Manifesto. In *the Proceedings of the 5th Linux Expo, March 1999, in Raleigh North Carolina*

[4] Volkmar Uhlig, Joshua LeVasseur, Espen Skoglund, and Uwe Dannowski. Towards Scalable Multiprocessor Virtual Machines. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*

[5] Philip M. Wells, Koushik Chakraborty, and Gurindar S. Sohi. Hardware Support for Spin Management in Overcommitted Virtual Machines. In Proc. *of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT-2006),* Sept. 2006, Seattle, WA

[6] J. K. Ousterhout. Scheduling Techniques for Concurrent Systems. *Proceedings of Third International Conference on Distributed Computing Systems, 1982*

[7] VMware, Inc. VMware vSphere(TM) 4: The CPU Scheduler. in VMware(R) ESX(TM) 4 http://www.vmware.com/files/pdf/perf-vsphere-cpu_scheduler.pdf